Getting Started

What‘s Inside this Course?

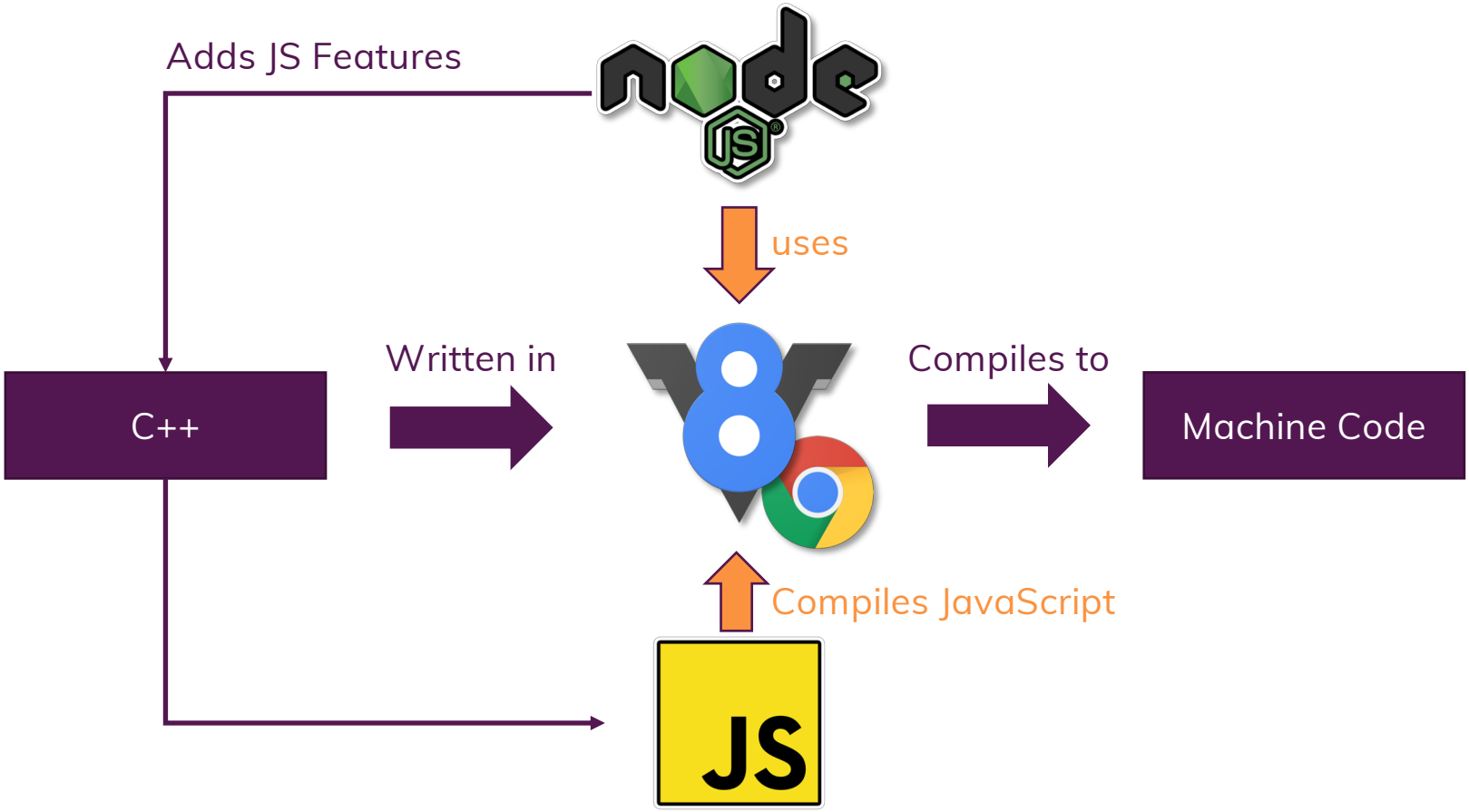
ACADEMIND

What is Node.js?

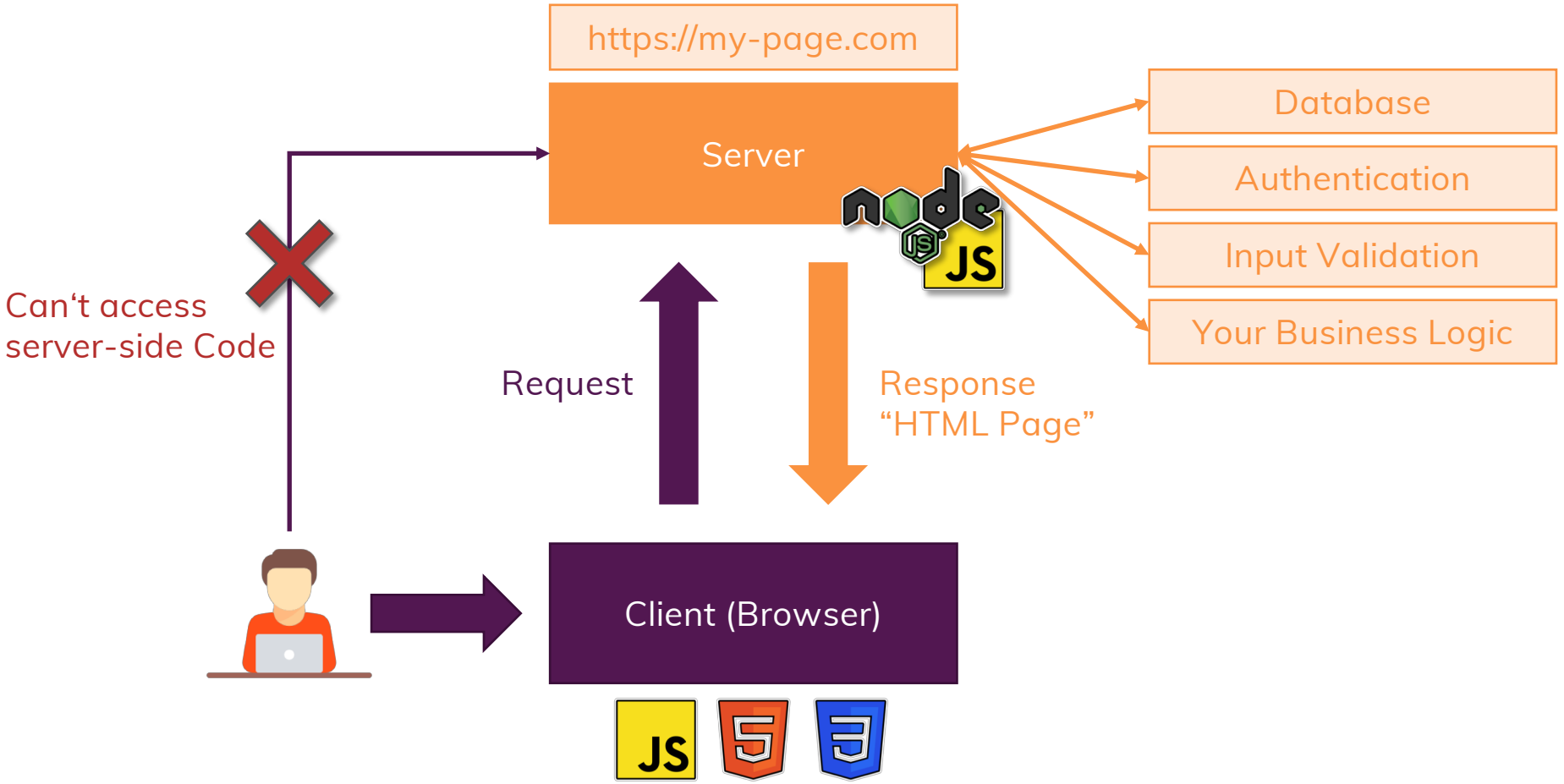
A JavaScript Runtime

"JavaScript on the Server"

What Does That Mean?

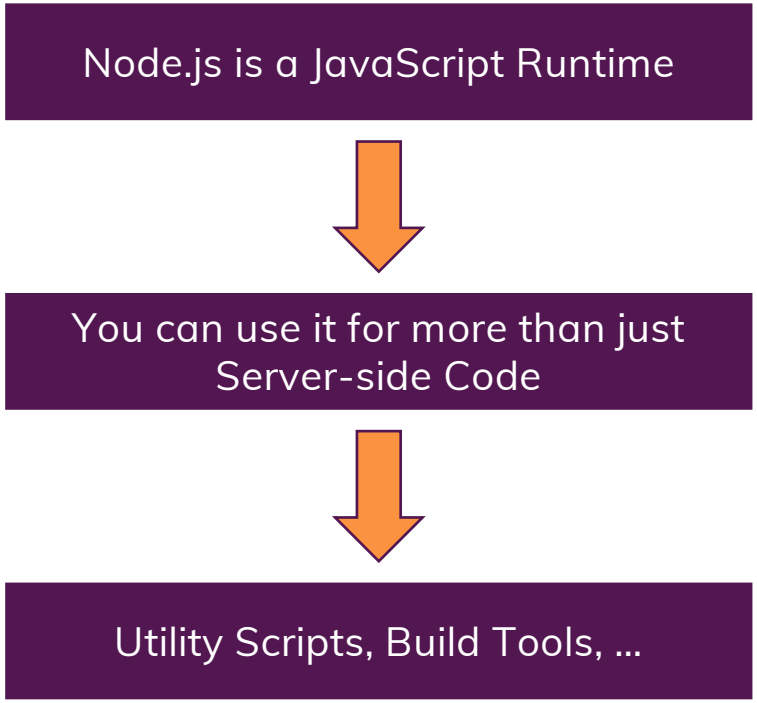


JavaScript on the Server

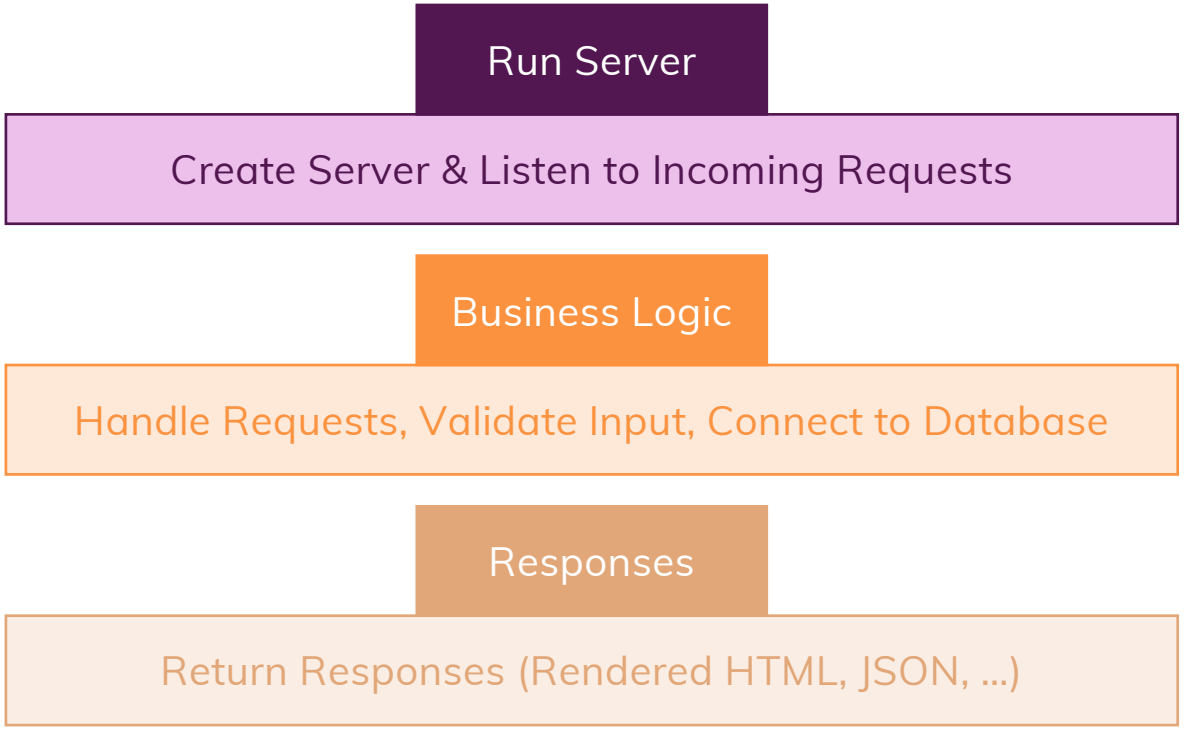


ACADEMIND

Side note: You're not limited to the Server!

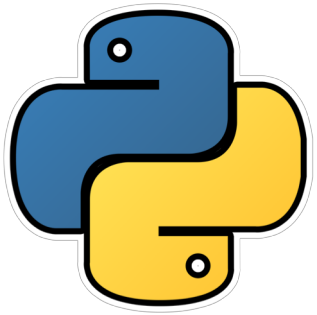


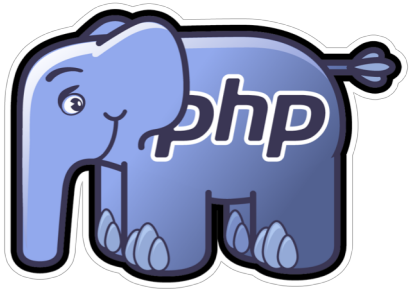
Node.js' Role (in Web Development)


Run Server
Create Server & Listen to Incoming Requests
Business Logic
Handle Requests, Validate Input, Connect to Database
Responses
Return Responses (Rendered HTML, JSON, ...)

ACADEMIND

Alternatives





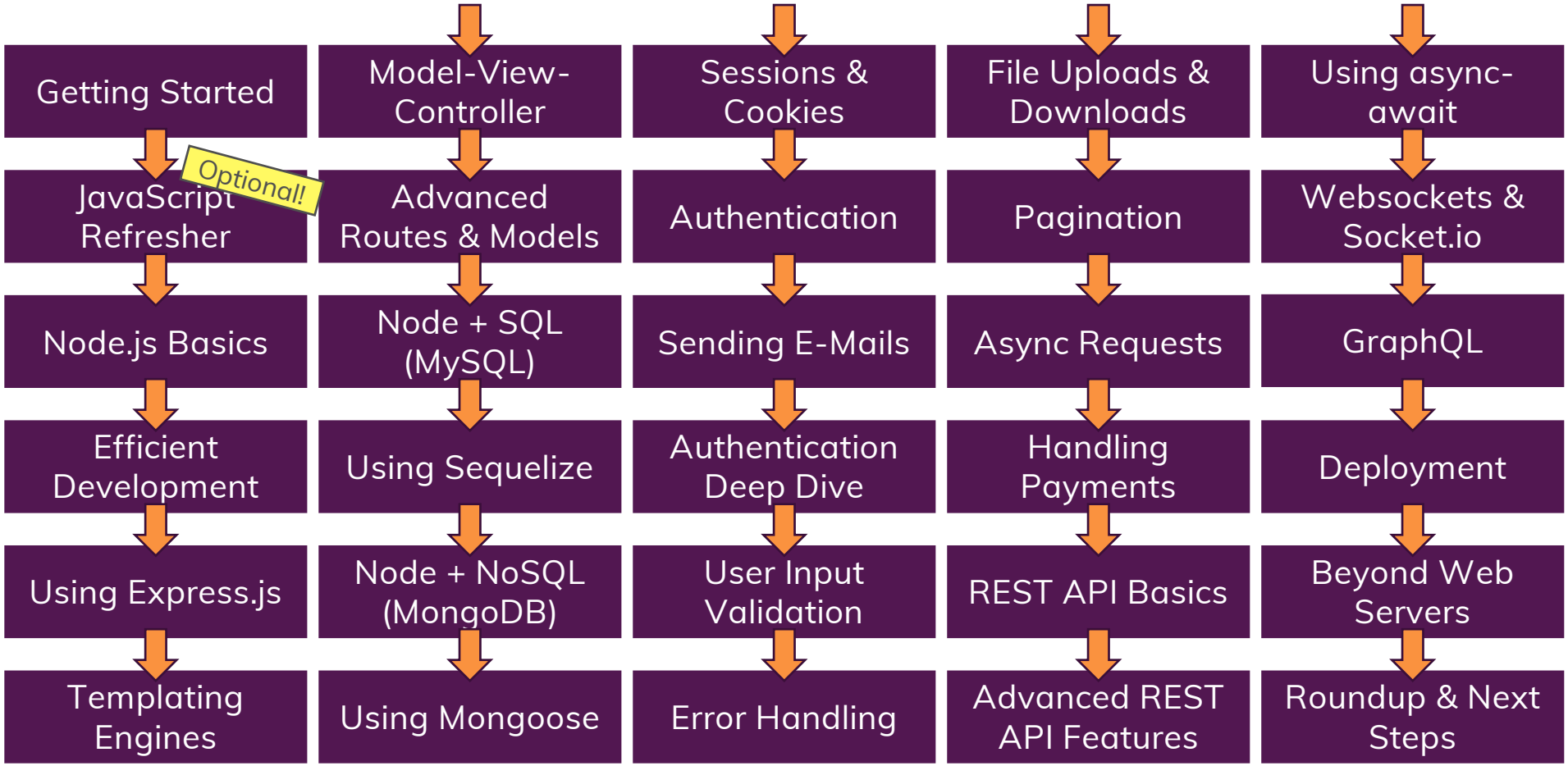
And More (Ruby, ASP.NET, ...)

Ultimately, you can build the same web apps with all of them

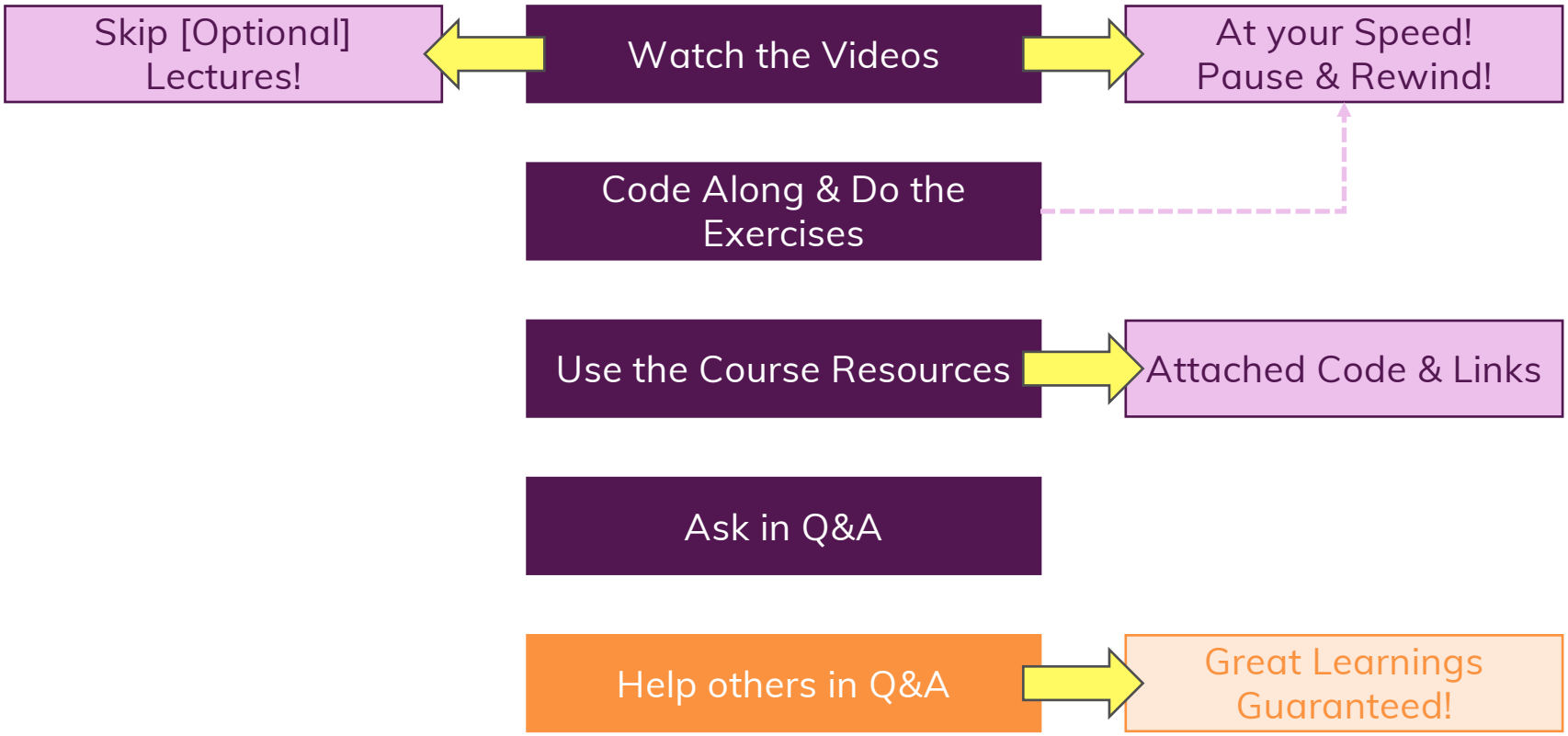
Choose your preferred syntax, community, ecosystem

ACADEMIND

Course Outline

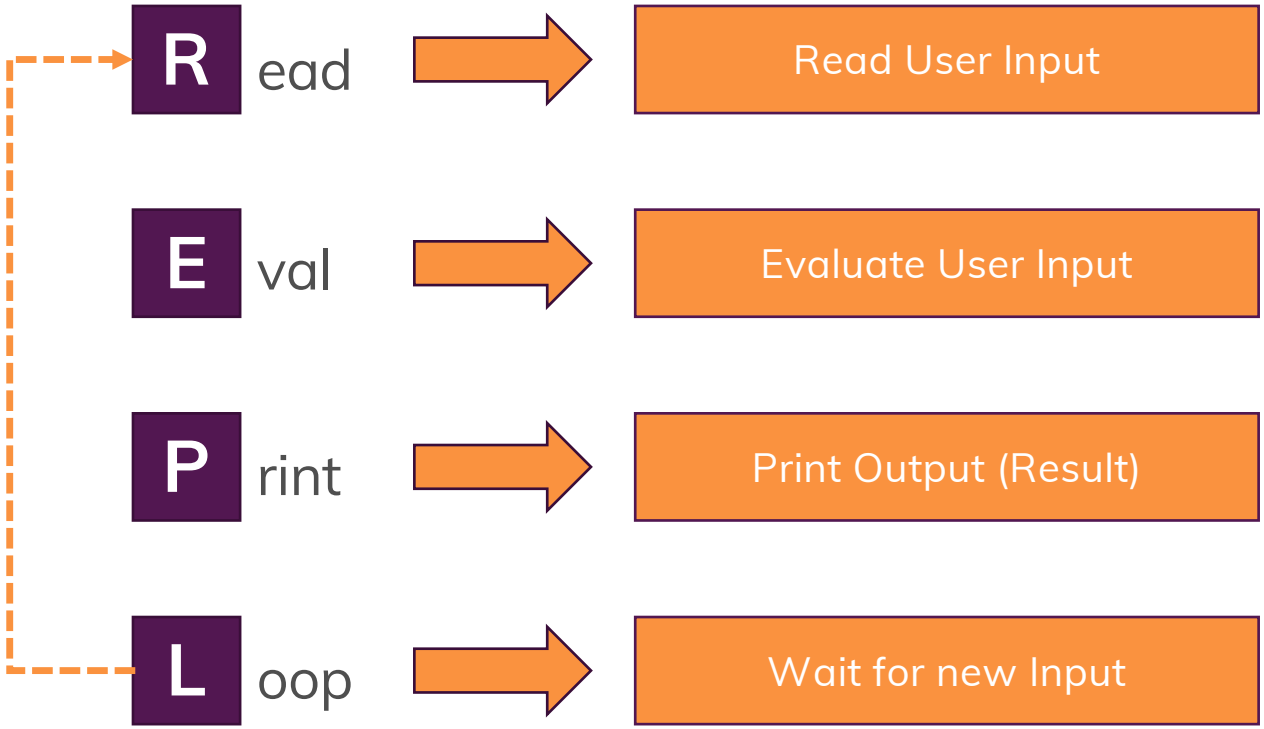


How To Get The Most Out Of The Course


Skip [Optional] Lectures!
Watch the Videos
At your Speed! Pause & Rewind!
Code Along & Do the Exercises
Use the Course Resources
Attached Code & Links
Ask in Q&A
Help others in Q&A
Great Learnings Guaranteed!

ACADEMIND

The REPL



Running Node.js Code

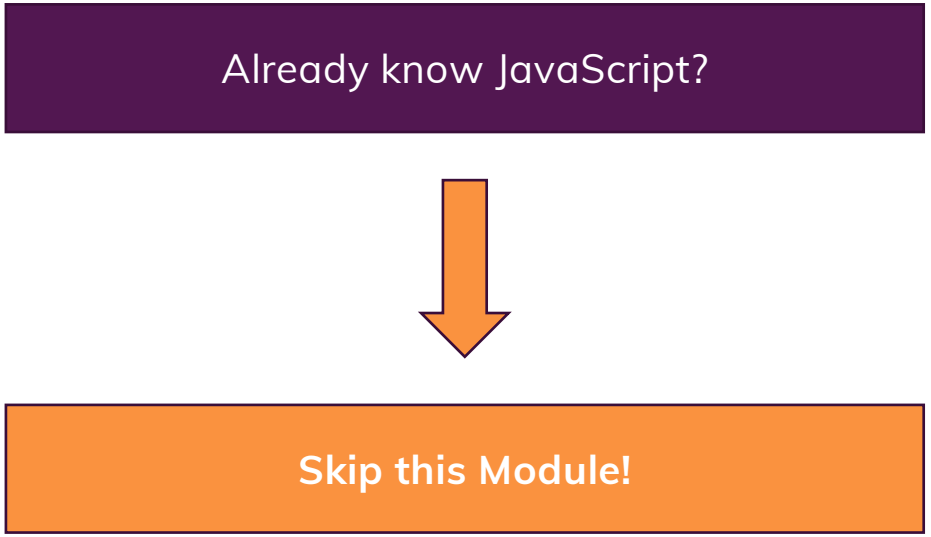
Execute Files	Use the REPL
Used for real apps	Great playground!
Predictable sequence of steps	Execute code as you write it

ACADEMIND

JavaScript Basics

A Quick Refresher

Skip This Module!

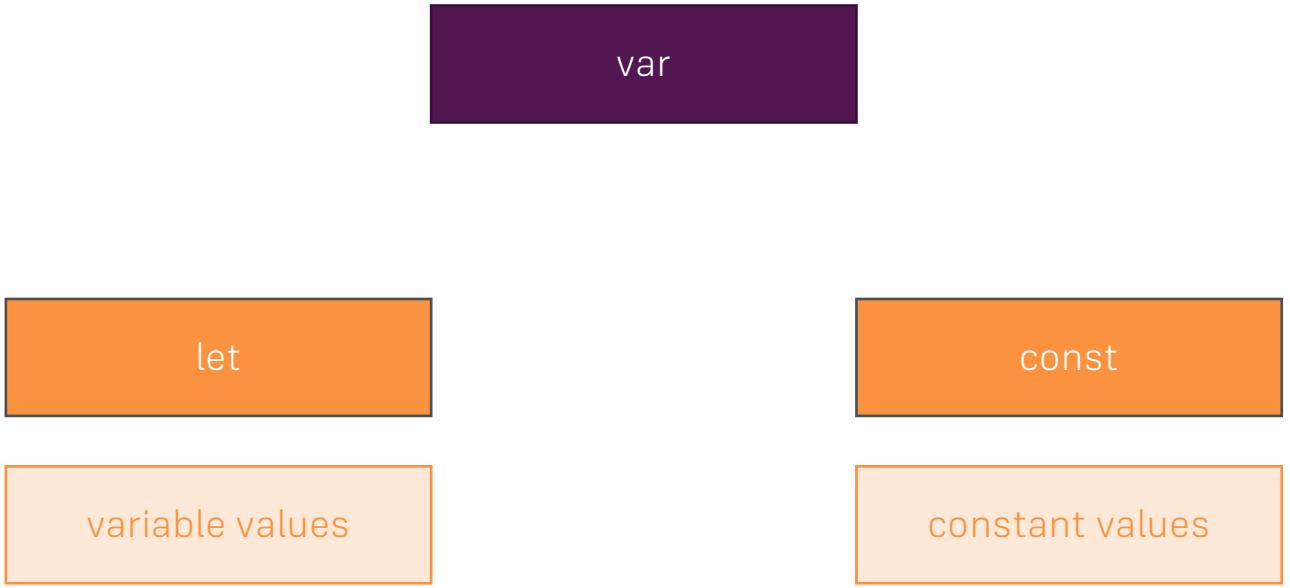


JavaScript Summary

Weakly Typed Language	Object-Oriented Language	Versatile Language
No explicit type assignment	Data can be organized in logical objects	Runs in browser & directly on a PC/ server
Data types can be switched dynamically	Primitive and reference types	Can perform a broad variety of tasks

ACADEMIND

let & const



ACADEMIND

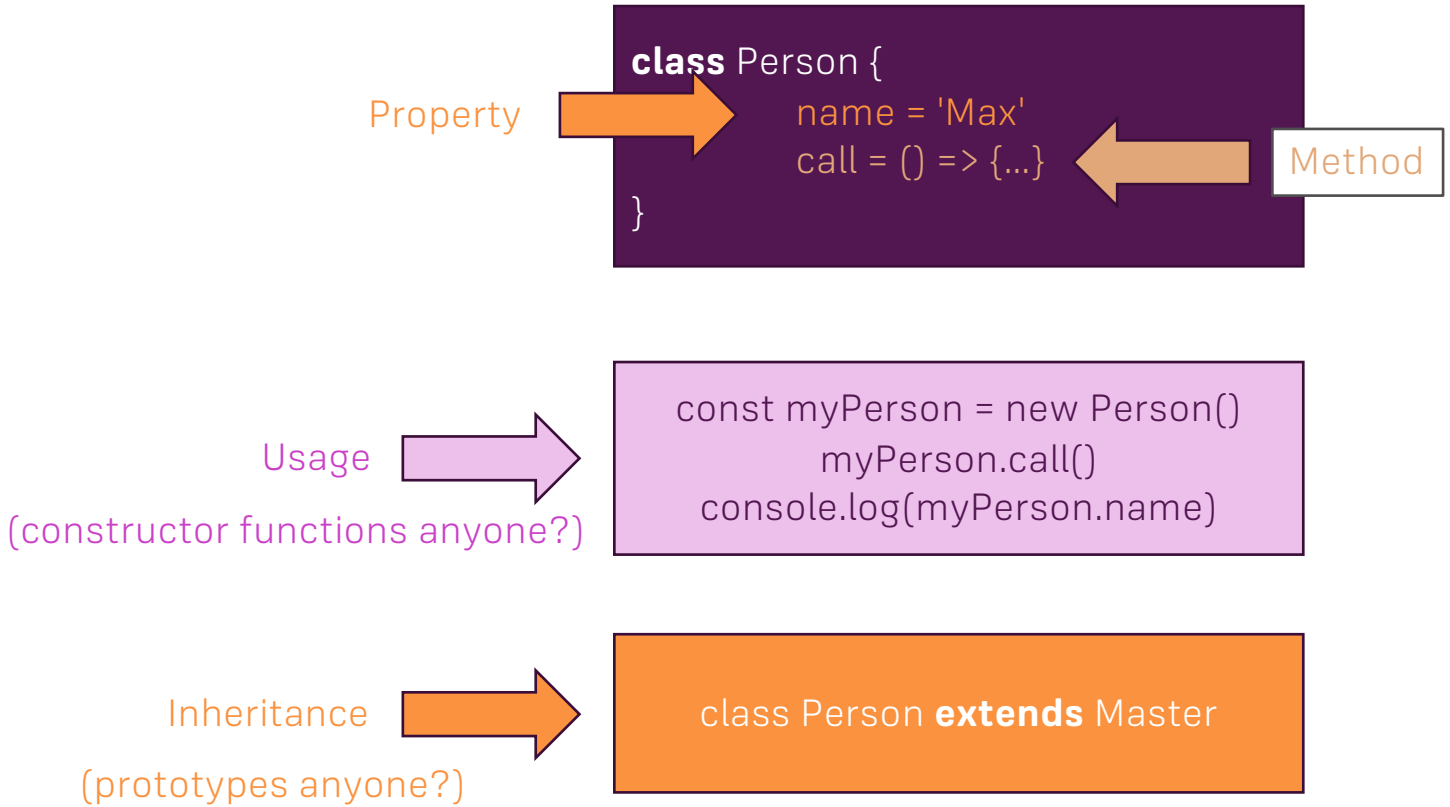
Arrow Functions

```
function myFnc() {
  ...
}
```

```
const myFnc = () => {
  ...
}
```

No more issues with the this keyword!

Classes


Property
class Person {
name = 'Max'
call = () => {...}
}
Method
Usage
(constructor functions anyone?)
const myPerson = new Person()
myPerson.call()
console.log(myPerson.name)
Inheritance
(prototypes anyone?)
class Person extends Master

Classes, Properties & Methods

Properties are like “variables attached to classes/ objects”

ES6

```
constructor () {
        this.myProperty = ' value'
}
```

ES7

```
myProperty = ' value'
```

Methods are like “functions attached to classes/ objects”

ES6

```
myMethod () { ... }
```

ES7

```
myMethod = () => { ... }
```

Spread & Rest Operators

...

Spread

Used to split up array elements OR object properties

```
const newArray = [...oldArray, 1, 2]
const newObject = { ...oldObject, newProp: 5 }
```

Rest

Used to merge a list of function arguments into an array

```
function sortArgs(...args) {
    return args.sort()
}
```

Destructuring

Easily extract array elements or object properties and store them in variables

Array Destructuring

```
[a, b] = ['Hello', 'Max']
console.log(a) // Hello
console.log(b) // Max
```

Object Destructuring

```
{name} = {name: 'Max', age: 28}
console.log(name) // Max
console.log(age) // undefined
```

ACADEMIND

Node.js Basics

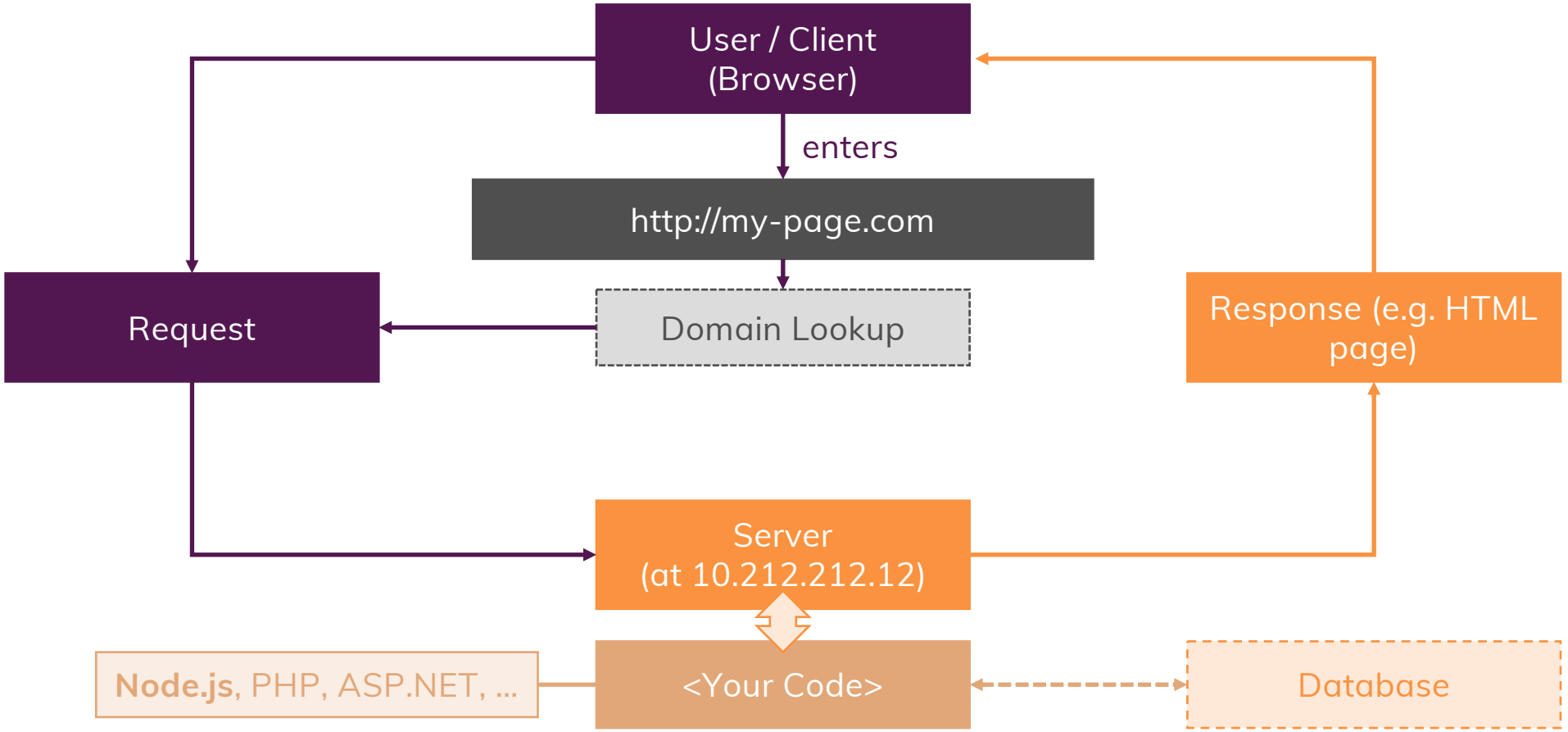
The Essential Knowledge You Need

ACADEMIND

What's In This Module?

- How Does The Web Work (Refresher)?
- Creating a Node.js Server
- Using Node Core Modules
- Working with Requests & Responses (Basics)
- Asynchronous Code & The Event Loop

How the Web Works



HTTP, HTTPS

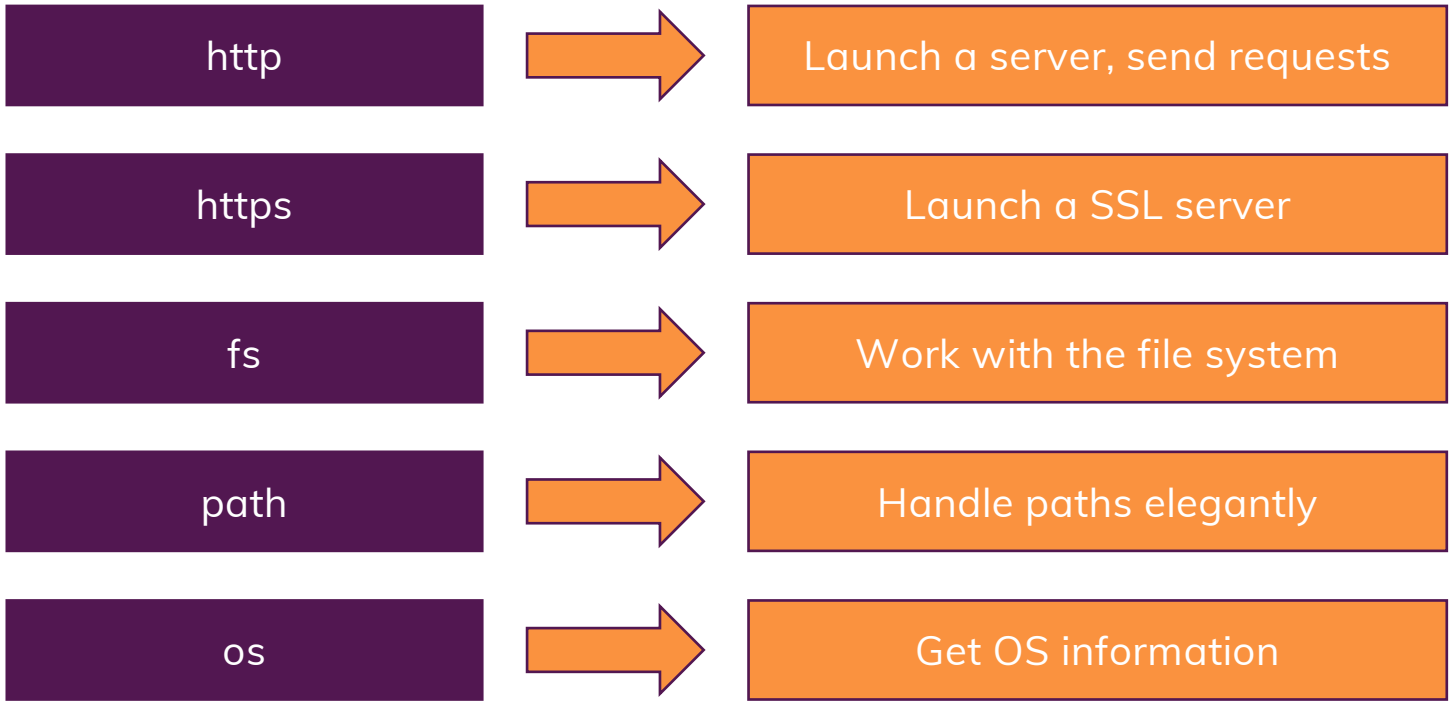
Hyper Text Transfer Protocol

A Protocol for Transferring Data which is understood by Browser and Server

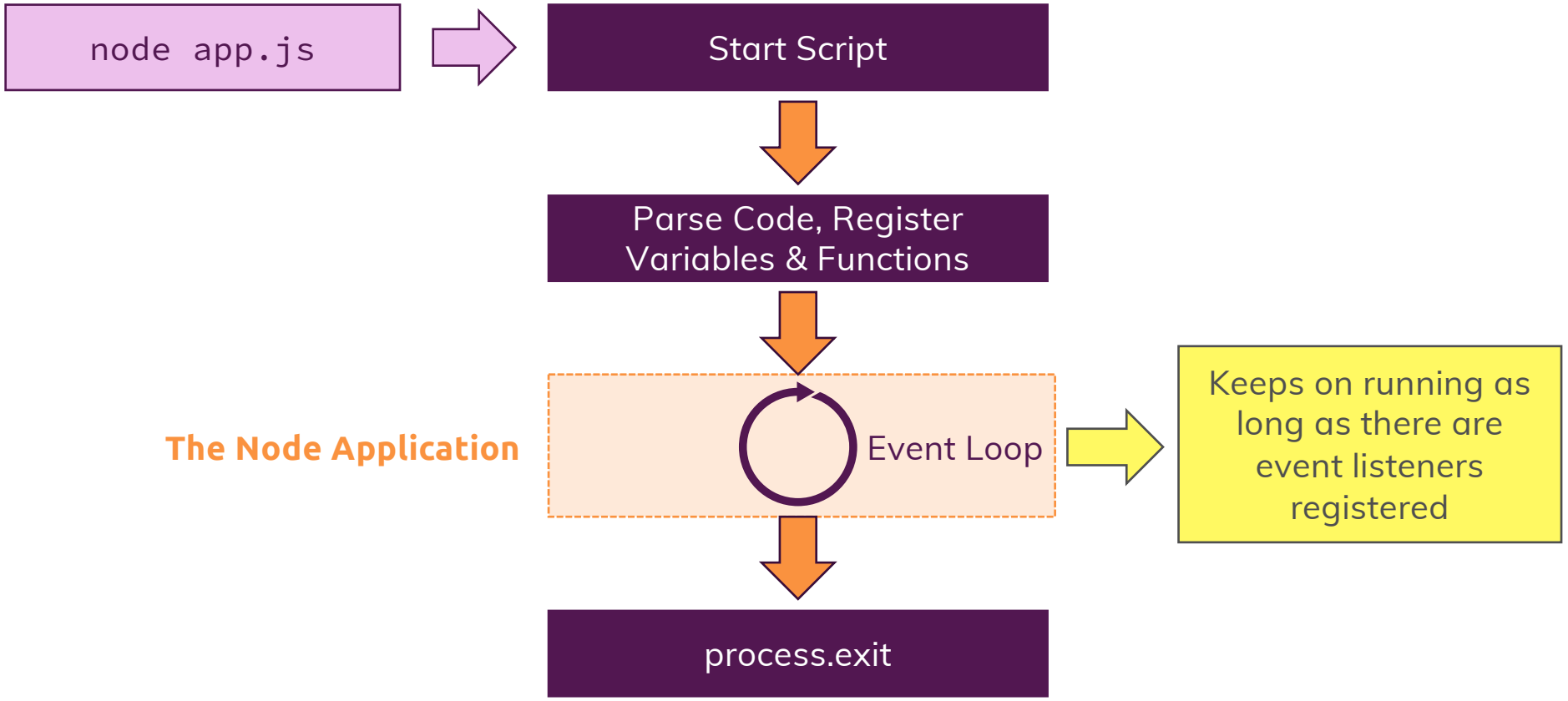
Hyper Text Transfer Protocol Secure

HTTP + Data Encryption (during Transmission)

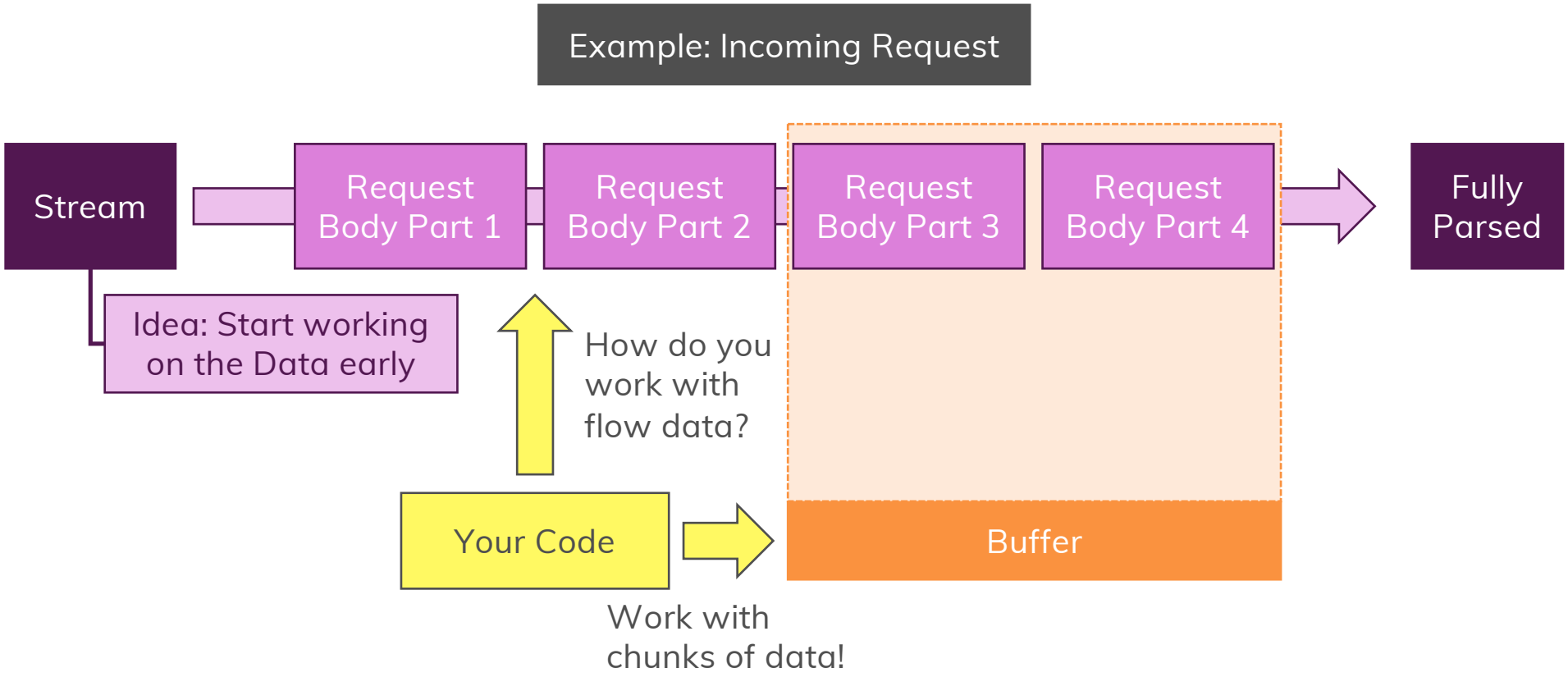
Core Modules



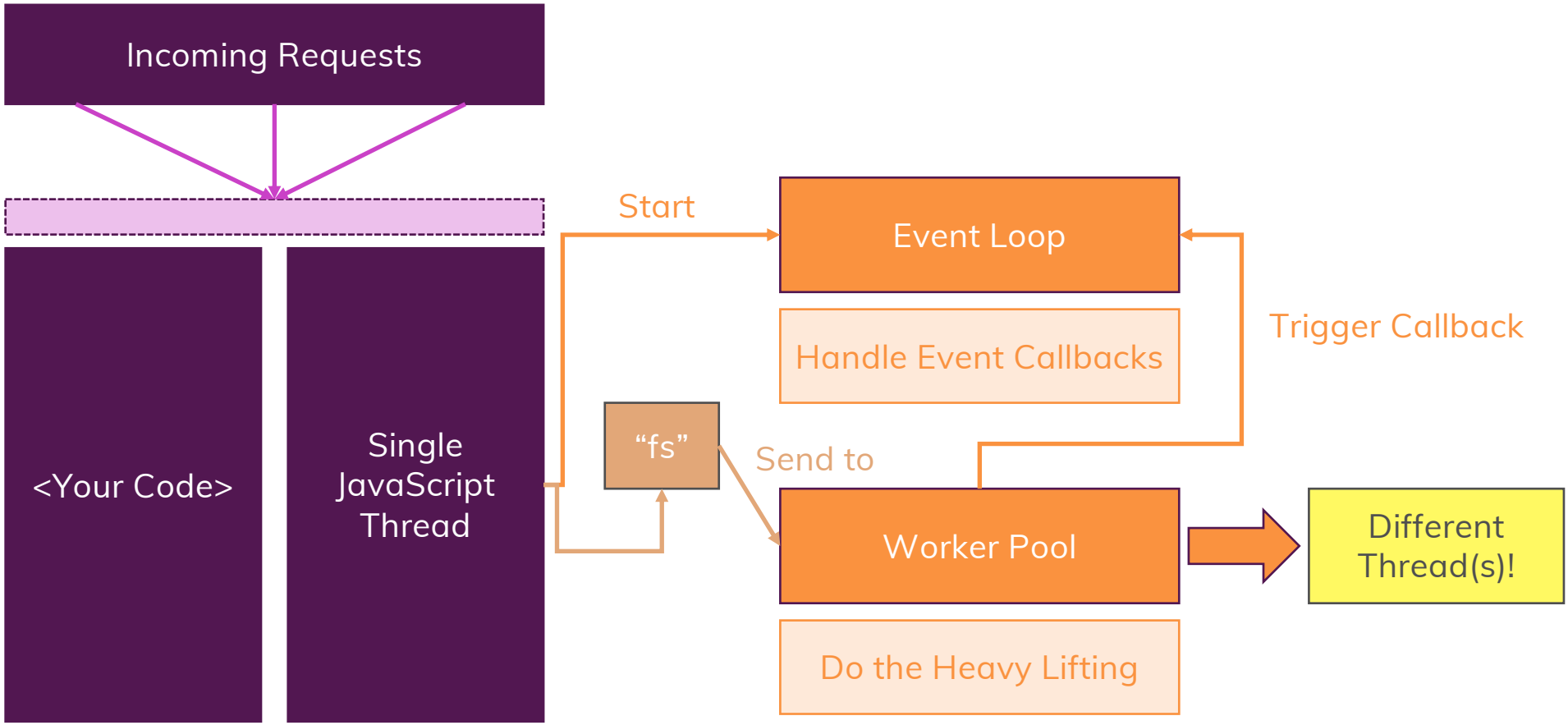
Node.js Program Lifecycle



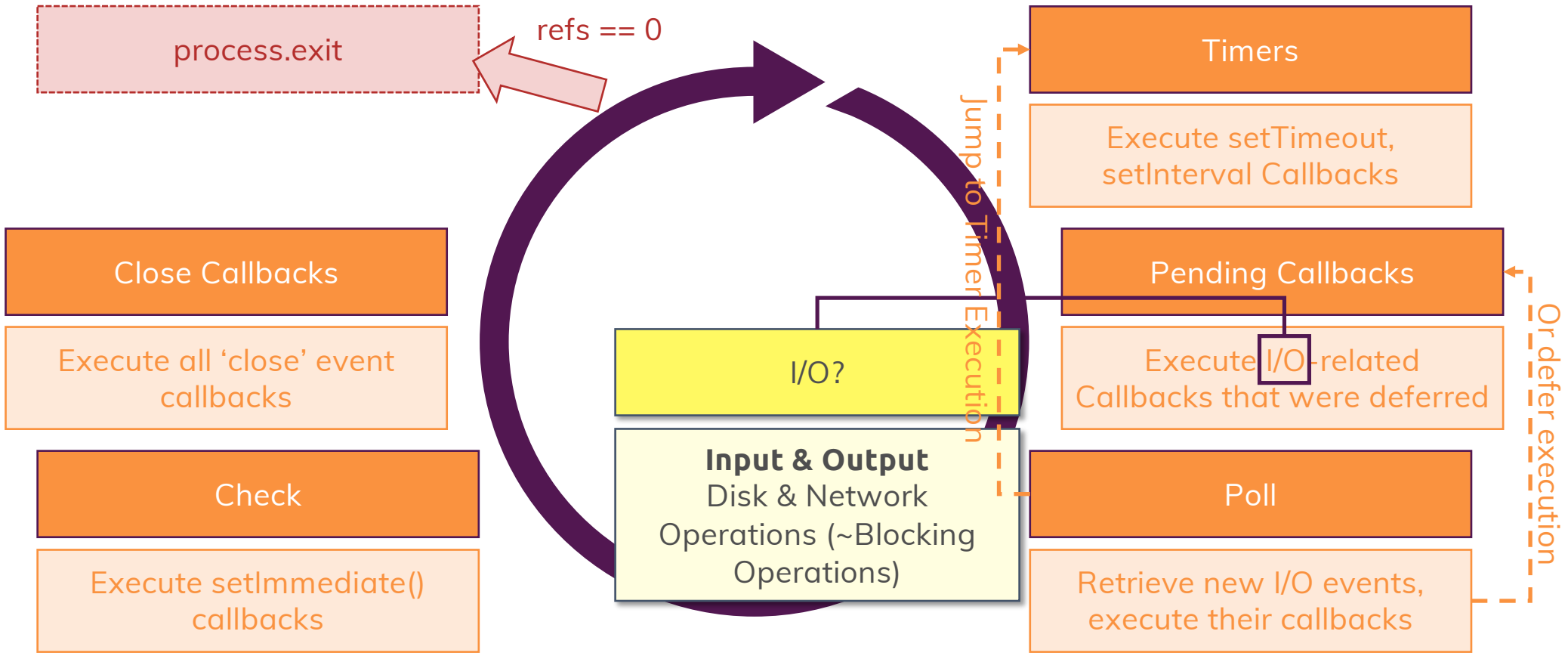
Streams & Buffers



Single Thread, Event Loop & Blocking Code

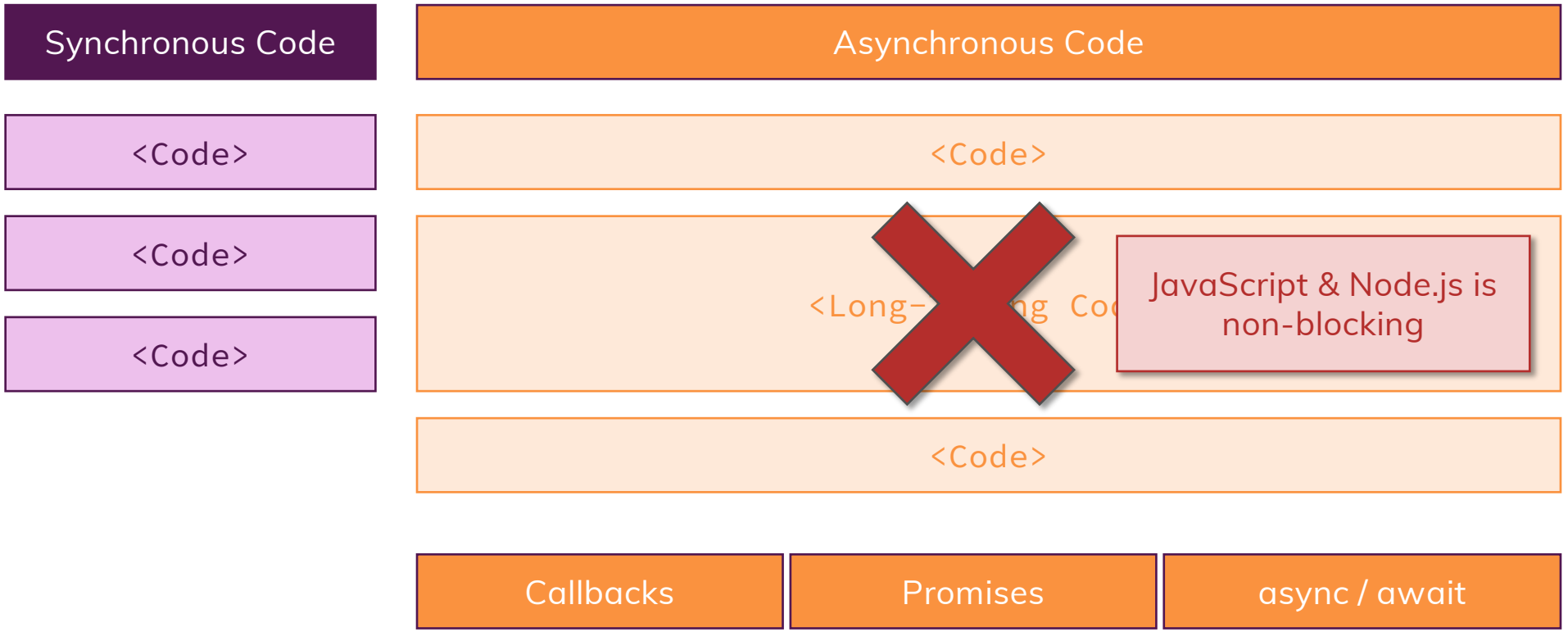


The Event Loop



ACADEMIND

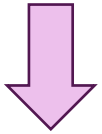
Asynchronous Code



Node's Module System

`module.exports = …`

`exports` is an alias for `module.exports`



`exports.something = …`

But `exports = { ... }` won't work! Because you then re-assign the alias!

Module Summary

How the Web Works

Client => Request => Server => Response => Client

Program Lifecycle & Event Loop

- Node.js runs non-blocking JS code and uses an event-driven code (”Event Loop”) for running your logic
- A Node program exits as soon as there is no more work to do
- Note: The createServer() event never finishes by default

Asynchronous Code

- JS code is non-blocking
- Use callbacks and events => Order changes!

Requests & Responses

- Parse request data in chunks (Streams & Buffers)
- Avoid “double responses”

Node.js & Core Modules

- Node.js ships with multiple core modules (http, fs, path, ...)
- Core modules can be imported into any file to be used there
- Import via require(‘module’)

The Node Module System

- Import via require(‘./path-to-file’) for custom files or require(‘module’) for core & third-party modules
- Export via module.exports or just exports (for multiple exports

Assignment

1	Spin up a Node.js-driven Server (on port 3000)
2	Handle two Routes: “/” and “/users” ▪ Return some greeting text on “/” ▪ Return a list of dummy users (e.g. User 1)
3	Add a form with a “username” <input> to the “/” page and submit a POST request to “/create-user” upon a button click
4	Add the “/create-user” route and parse the incoming data (i.e. the username) and simply log it to the console

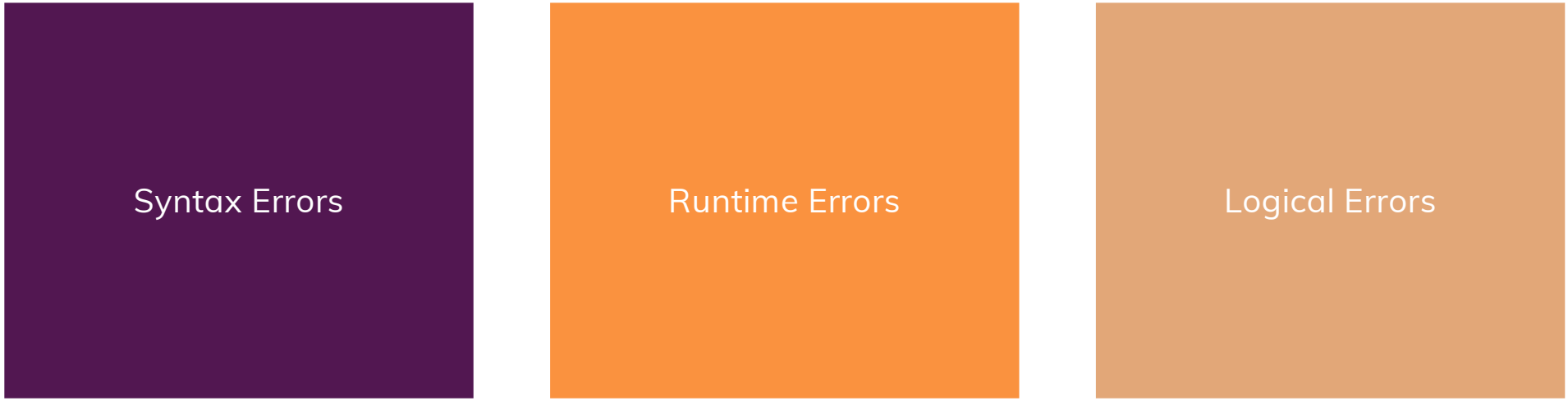
ACADEMIND

Debugging & Easier Development

Fixing Errors, Developing Efficiently

ACADEMIND

Types of Errors



ACADEMIND

Module Summary

npm

- npm stands for “Node Package Manager” and it allows you to manage your Node project and its dependencies
- You can initialize a project with `npm init`
- npm scripts can be defined in the package.json to give you “shortcuts” to common tasks/ commands

3rd Party Packages

- Node projects typically don’t just use core modules and custom code but also third-party packages
- You install them via npm
- You can differentiate between production dependencies (--save), development dependencies (--save-dev) and global dependencies (-g)

Types of Errors

- Syntax, runtime and logical errors can break your app
- Syntax and runtime errors throw (helpful) error messages (with line numbers!)
- Logical errors can be fixed with testing and the help of the debugger

Debugging

- Use the VS Code Node debugger to step into your code and go through it step by step
- Analyze variable values at runtime
- Look into (and manipulate) variables at runtime
- Set breakpoints cleverly (i.e. respect the async/ event-driven nature)

ACADEMIND

Express.js

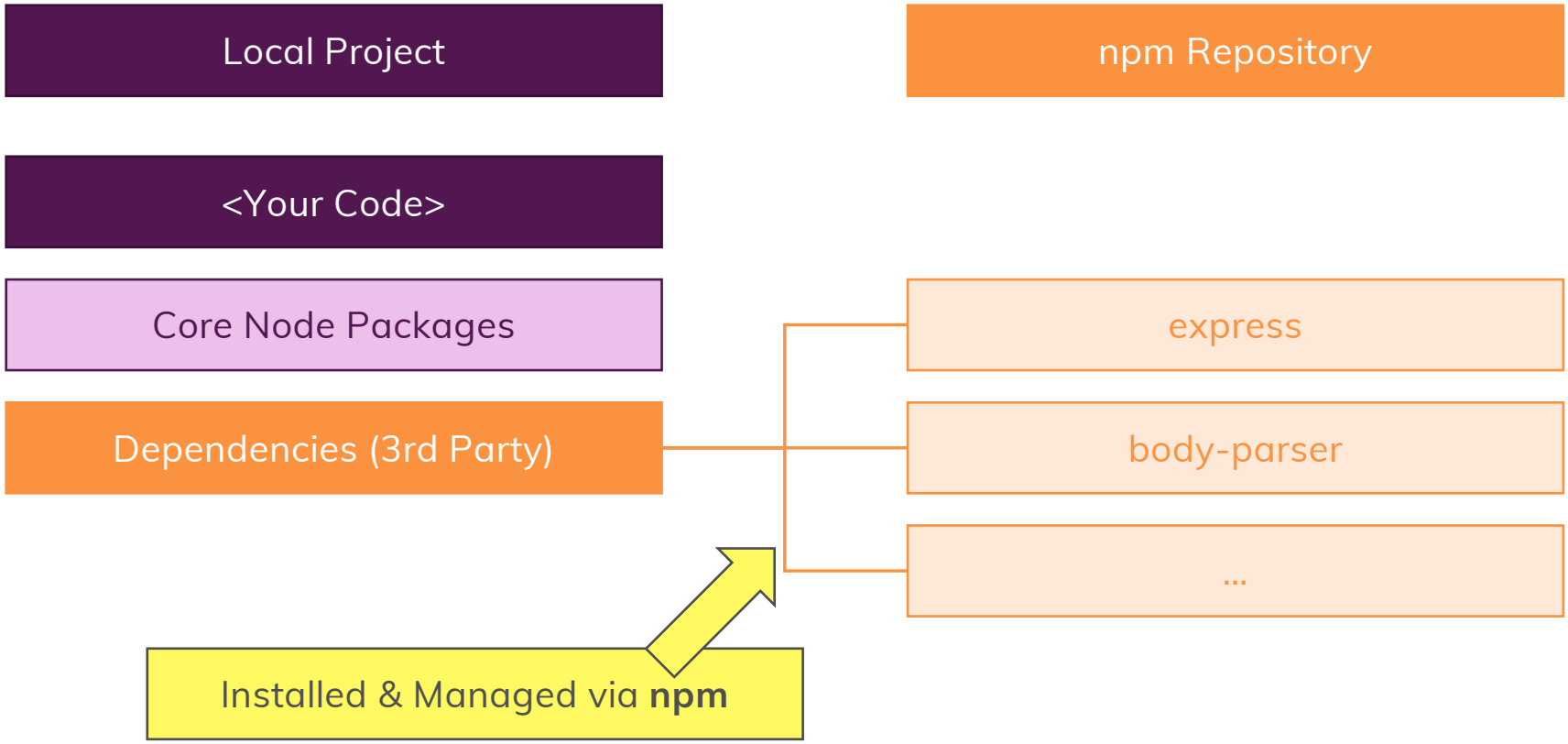
Don’t re-invent the Wheel!

ACADEMIND

What's In This Module?

- What is Express.js?
- Using Middleware
- Working with Requests & Responses (Elegantly!)
- Routing
- Returning HTML Pages (Files)

npm & Packages



ACADEMIND

What and Why?

Server Logic is Complex!

You want to focus on your Business Logic, not on the nitty-gritty Details!

Framework: Helper functions, tools & rules that help you build your application!

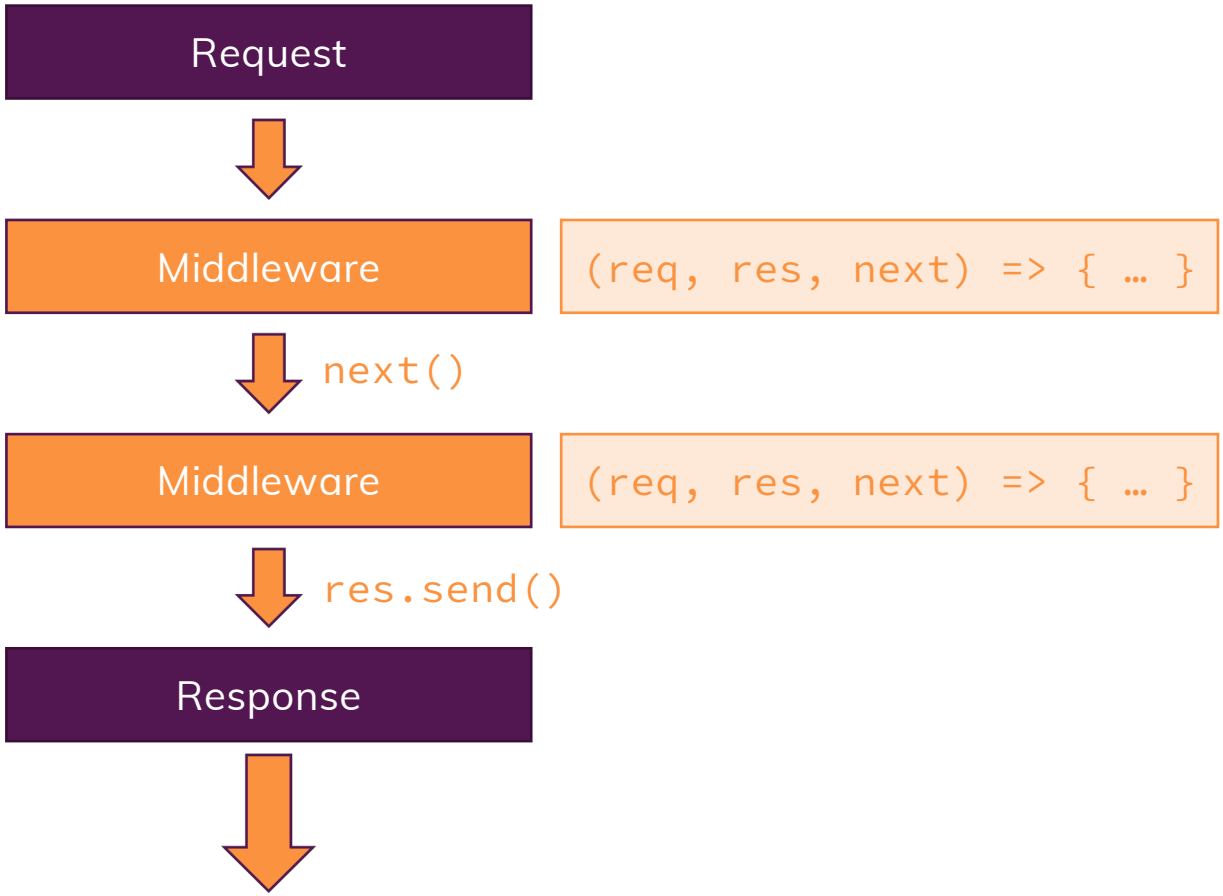
Use a Framework for the Heavy Lifting!

What Does Express.js Help You With?

Parsing Requests & Sending Responses	Routing	Managing Data
Extract Data	Execute different Code for different Requests	Manage Data across Requests (Sessions)
Render HTML Pages	Filter / Validate incoming Requests	Work with Files
Return Data / HTML Responses		Work with Databases

ACADEMIND

It's all about Middleware

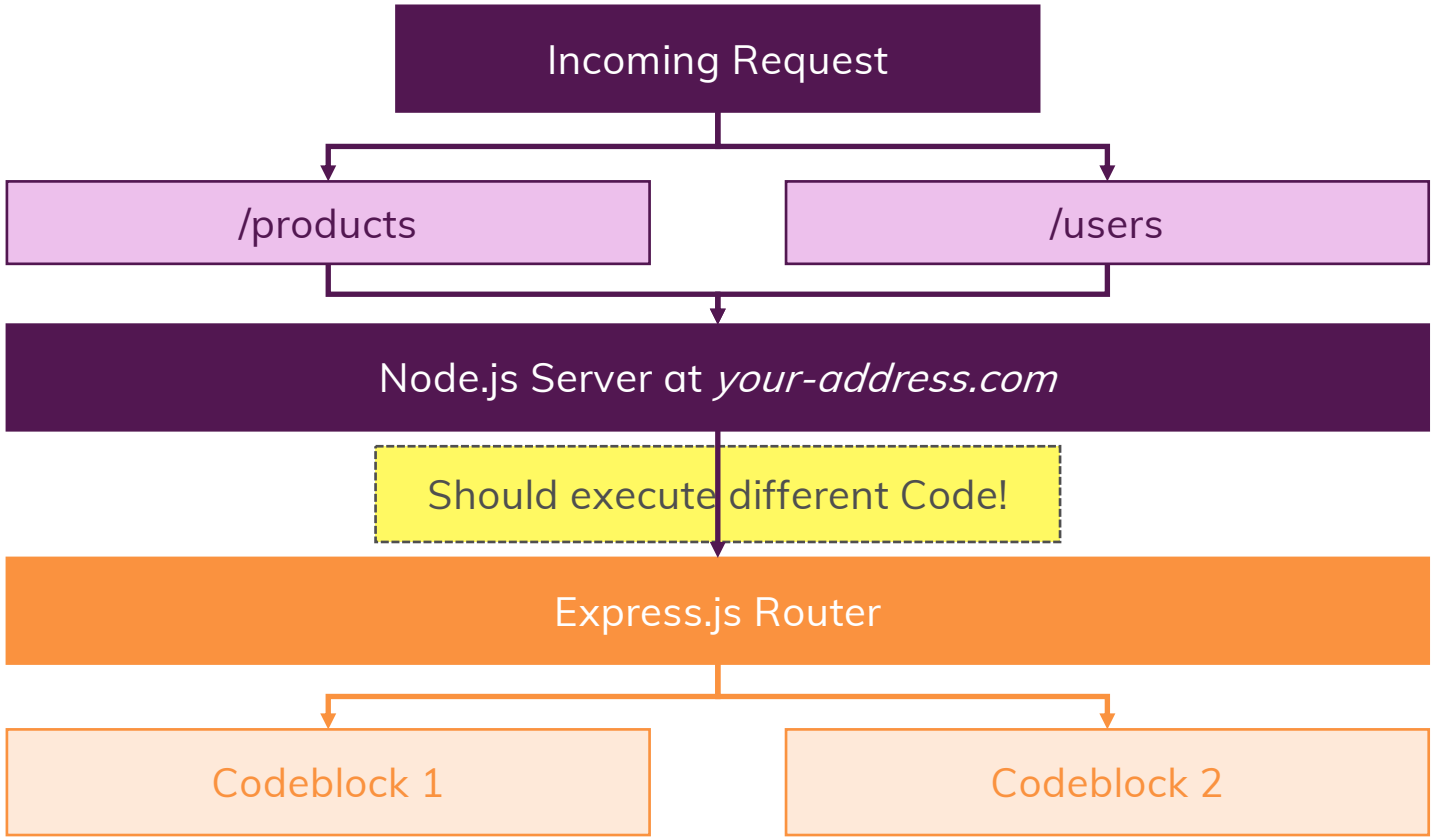


ACADEMIND

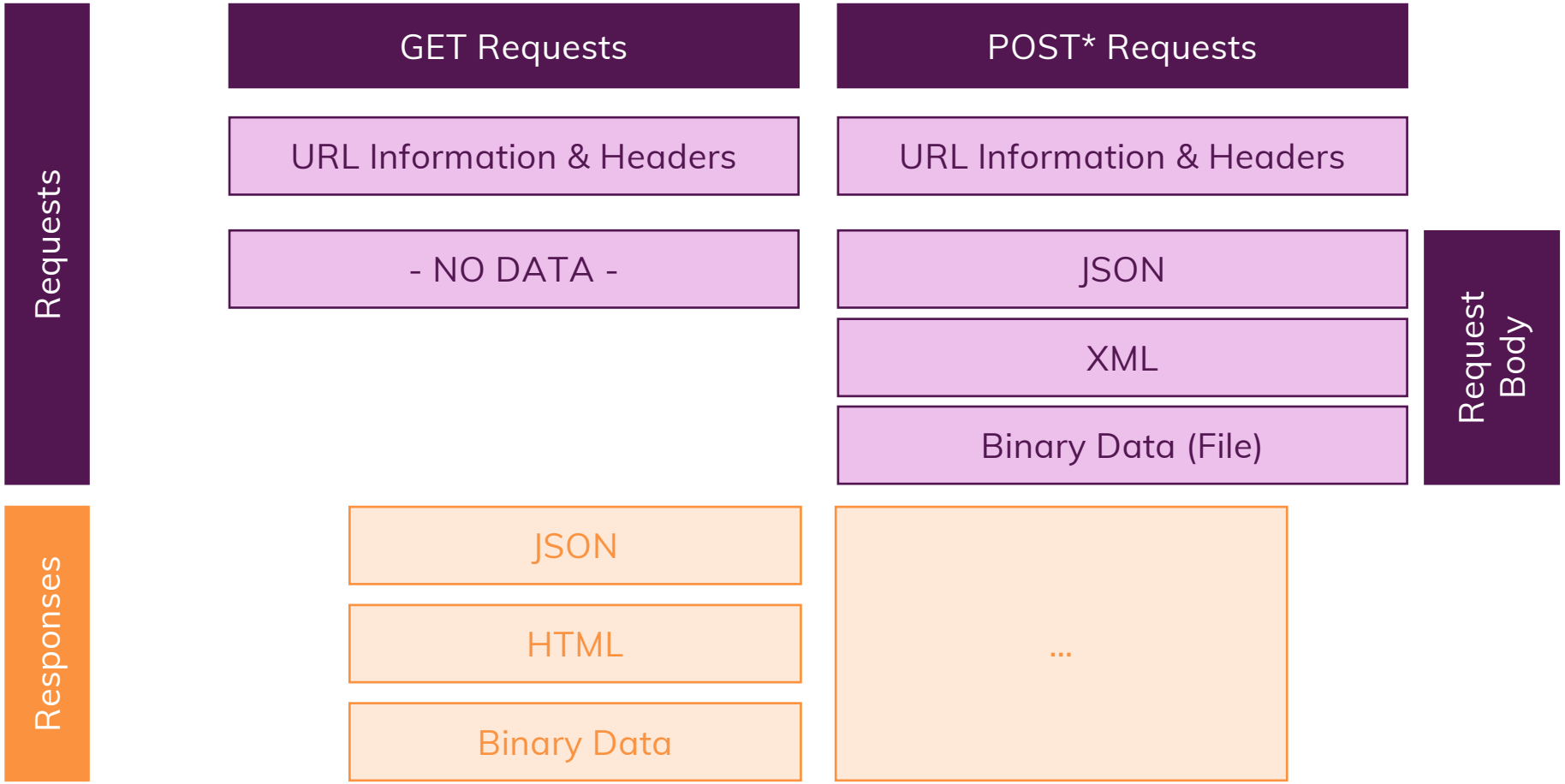
Assignment

1	Create a npm project and install Express.js (Nodemon if you want)
2	Create an Express.js app which funnels the requests through 2 middleware functions that log something to the console and return one response.
3	Handle requests to “/” and ”/users” such that each request only has one handler/ middleware that does something with it (e.g. send dummy response).

Routing


Incoming Request
/products
/users
Node.js Server at *your-address.com*
Should execute different Code!
Express.js Router
Codeblock 1
Codeblock 2

Request & Response Data



Alternatives to Express.js

- Vanilla Node.js
- Adonis.js
- Koa
- Sails.js
- ...

ACADEMIND

Assignment

1	Create a npm project and install Express.js (Nodemon if you want)
2	Create an Express.js app which serves two HTML files (of your choice/ with your content) for ”/” and “/users”.
3	Add some static (.js or .css) files to your project that should be required by at least one of your HTML files.

ACADEMIND

Module Summary

What is Express.js?

- Express.js is Node.js framework – a package that adds a bunch of utility functions and tools and a clear set of rules on how the app should be built (middleware!)
- It's highly extensible and other packages can be plugged into it (middleware!)

Routing

- You can filter requests by path and method
- If you filter by method, paths are matched exactly, otherwise, the first segment of a URL is matched
- You can use the express.Router to split your routes across files elegantely

Middleware, next() and res()

- Express.js relies heavily on middleware functions – you can easily add them by calling use()
- Middleware functions handle a request and should call next() to forward the request to the next function in line or send a response

Serve Files

- You're not limited to serving dummy text as a response
- You can sendFile()s to your users – e.g. HTML files
- If a request is directly made for a file (e.g. a .css file is requested), you can enable static serving for such files via express.static()

Dynamic Content & Templates

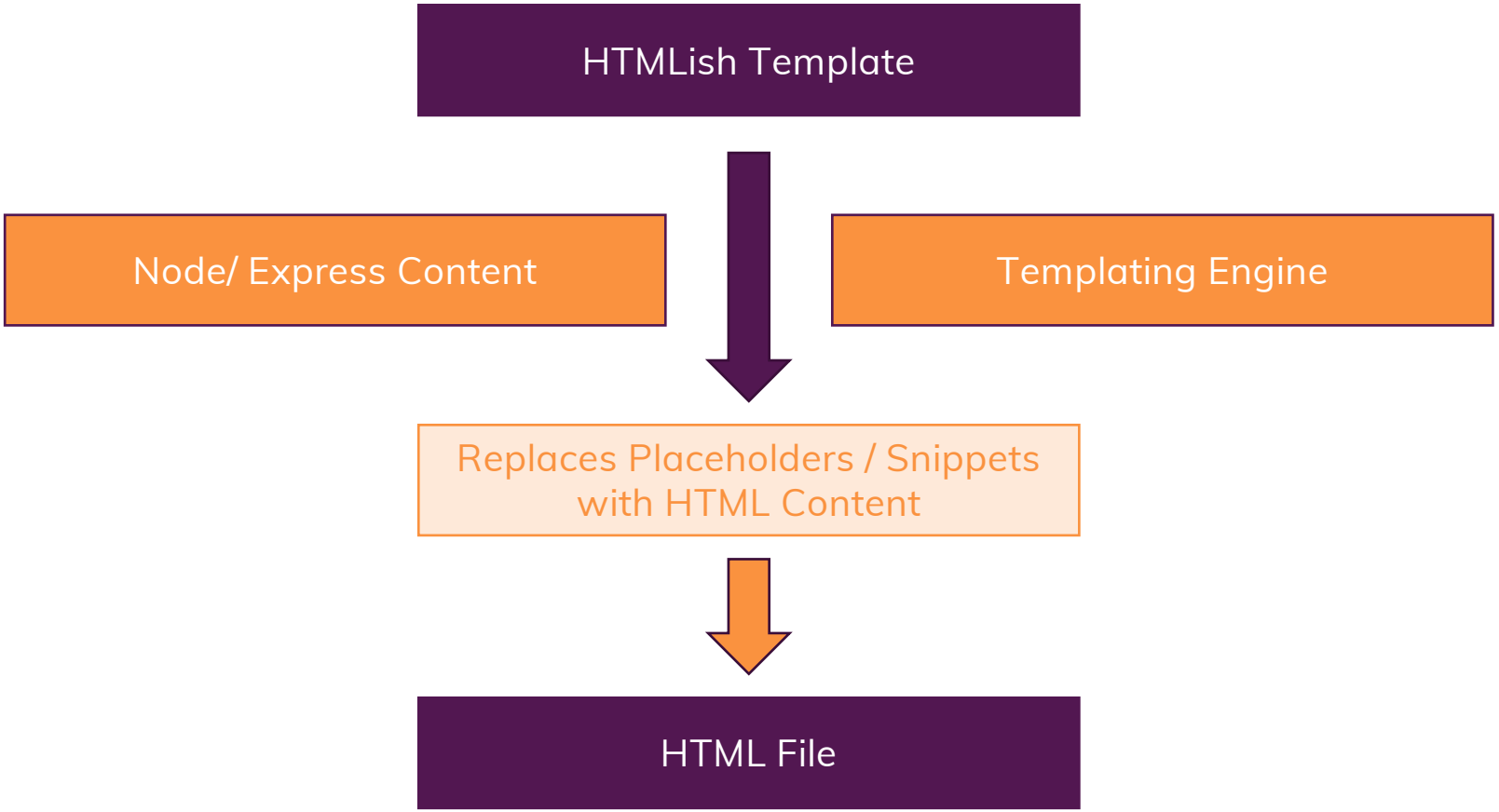
Rendering more than Static HTML

ACADEMIND

What's In This Module?

- Managing Data (without a Database)
- Render Dynamic Content in our Views
- Understanding Templating Engines

Templating Engines



ACADEMIND

Available Templating Engines

EJS	Pug (Jade)	Handlebars
`<p><%= name %></p>`	`p #{name}`	`<p>{{ name }}</p>`
Use normal HTML and plain JavaScript in your templates	Use minimal HTML and custom template language	Use normal HTML and custom template language

ACADEMIND

Assignment

1	Create a npm project and install Express.js (Nodemon if you want)
2	Add two routes: ▪ "/" => Holds a <form> that allows users to input their name ▪ "/users" => Outputs an with the user names (or some error text)

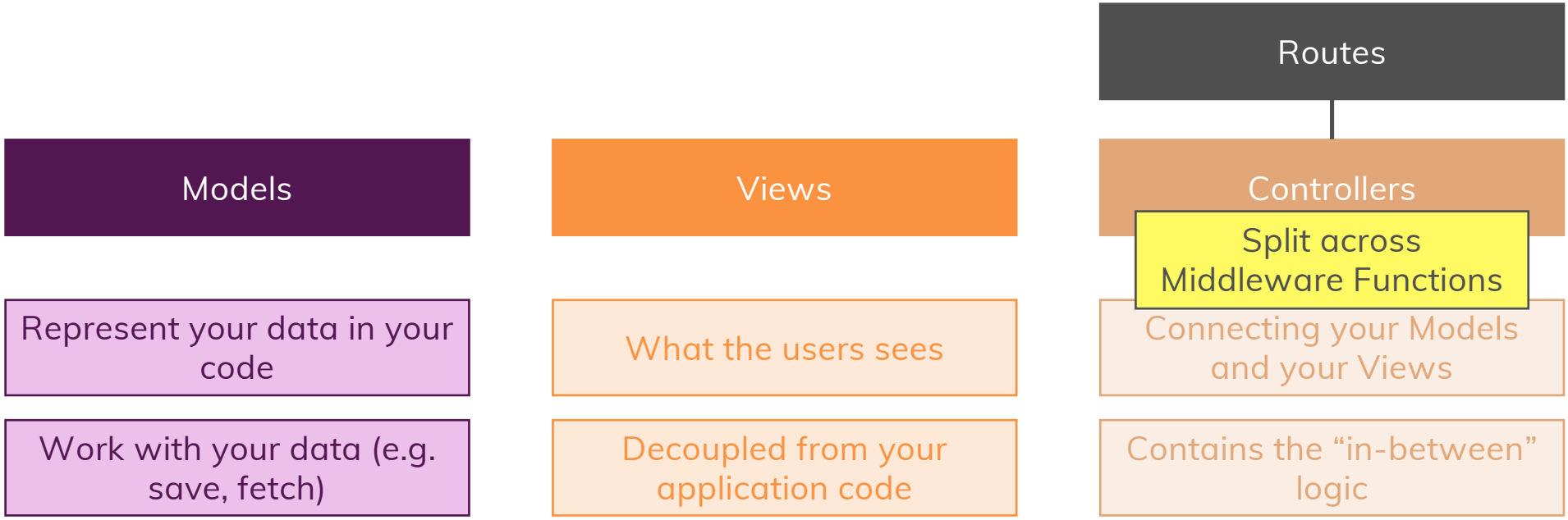
ACADEMIND

Model View Controller (MVC)

Structuring your Code

What's MVC?

Separation of Concerns



Module Summary

Model

- Responsible for representing your data
- Responsible for managing your data (saving, fetching, ...)
- Doesn't matter if you manage data in memory, files, databases
- Contains data-related logic

View

- What the user sees
- Shouldn't contain too much logic (Handlebars!)

Controller

- Connects Model and View
- Should only make sure that the two can communicate (in both directions)

ACADEMIND

Dynamic Routes & Advanced Models

Passing and Using Dynamic Data

ACADEMIND

What's In This Module?

- Passing Route Params
- Using Query Params
- Enhance our Models

Module Summary

Dynamic Routing

- You can pass dynamic path segments by adding a ":" to the Express router path
- The name you add after ":" is the name by which you can extract the data on `req.params`
- Optional (query) parameters can also be passed (`?param=value&b=2`) and extracted (`req.query.myParam`)

More on Models

- A Cart model was added – it holds static methods only
- You can interact between models (e.g. delete cart item if a product is deleted)
- Working with files for data storage is suboptimal for bigger amounts of data

ACADEMIND

Using Promises & Improving Async Code

Escaping Callback Hell

ACADEMIND

What's In This Module?

- What is Callback Hell?
- Promises to the Rescue!
- Writing Good Async Code

ACADEMIND

Callback Hell

```
fs.readFile('path', (err, fileContent) => {
       fs.writeFile('path', 'new content', (err) => {
              …
       })
});
```

Deeply nested callbacks can be hard to read, understand & maintain!

Module Summary

Dynamic Routing

- You can pass dynamic path segments by adding a ":" to the Express router path
- The name you add after ":" is the name by which you can extract the data on `req.params`
- Optional (query) parameters can also be passed (`?param=value&b=2`) and extracted (`req.query.myParam`)

More on Models

- A Cart model was added – it holds static methods only
- You can interact between models (e.g. delete cart item if a product is deleted)
- Working with files for data storage is suboptimal for bigger amounts of data

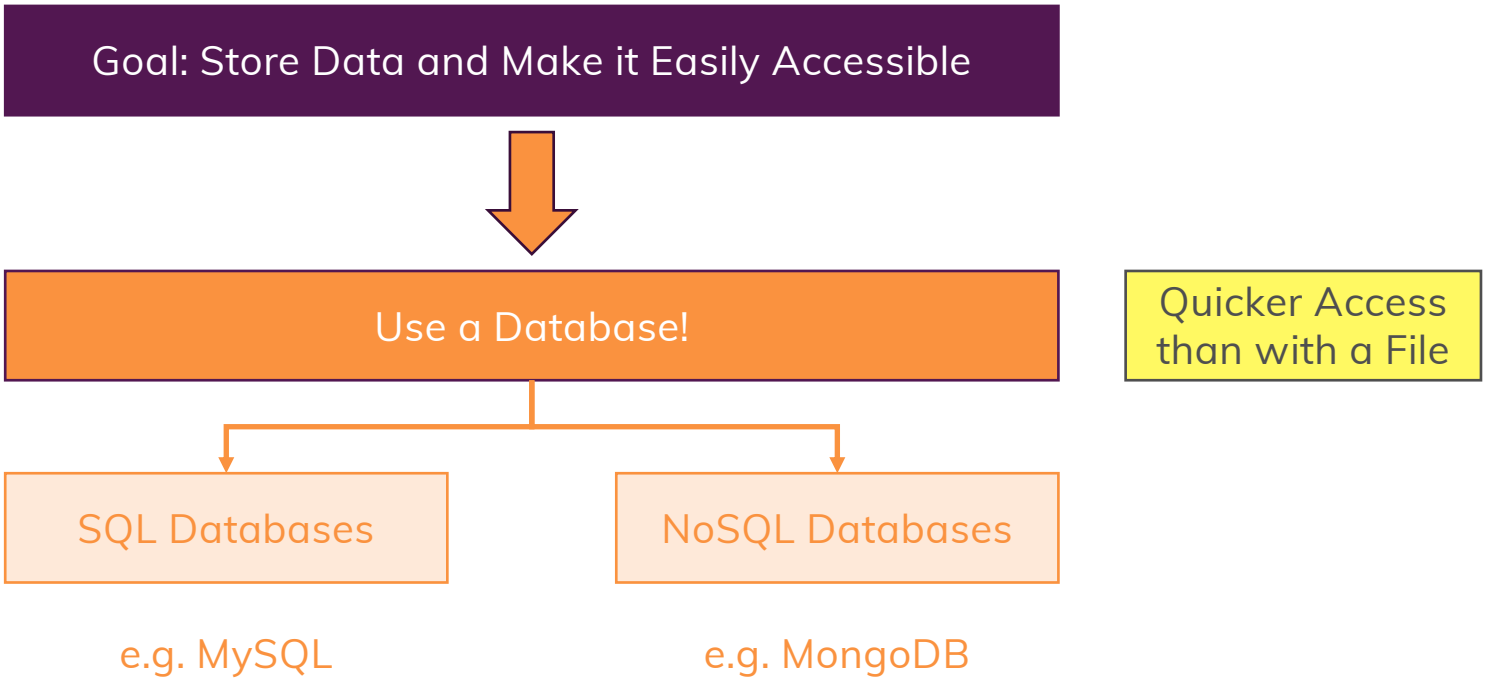
Storing Data in Databases

Storing Application Data Correctly

What's In This Module?

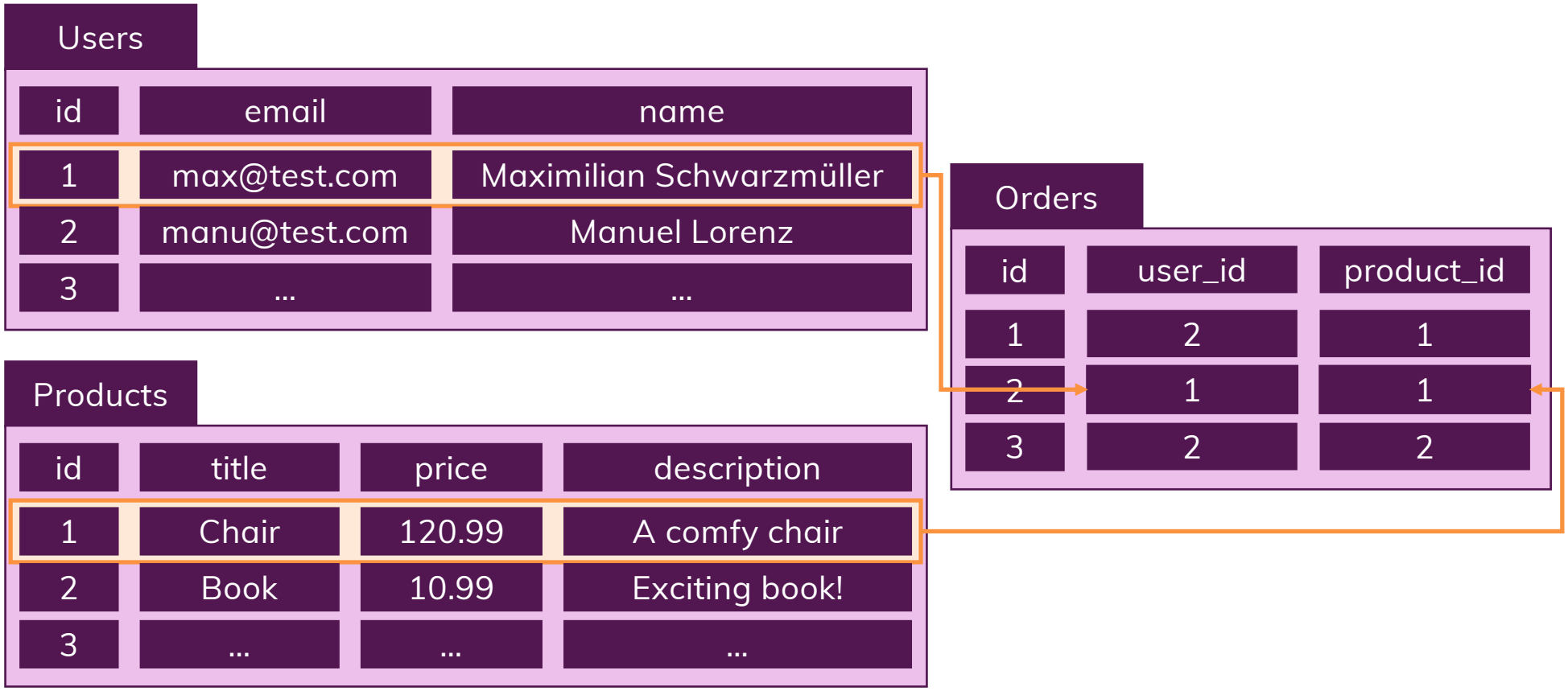
- Different Kinds of Databases (SQL vs NoSQL)
- Using SQL in a Node.js App

SQL vs NoSQL

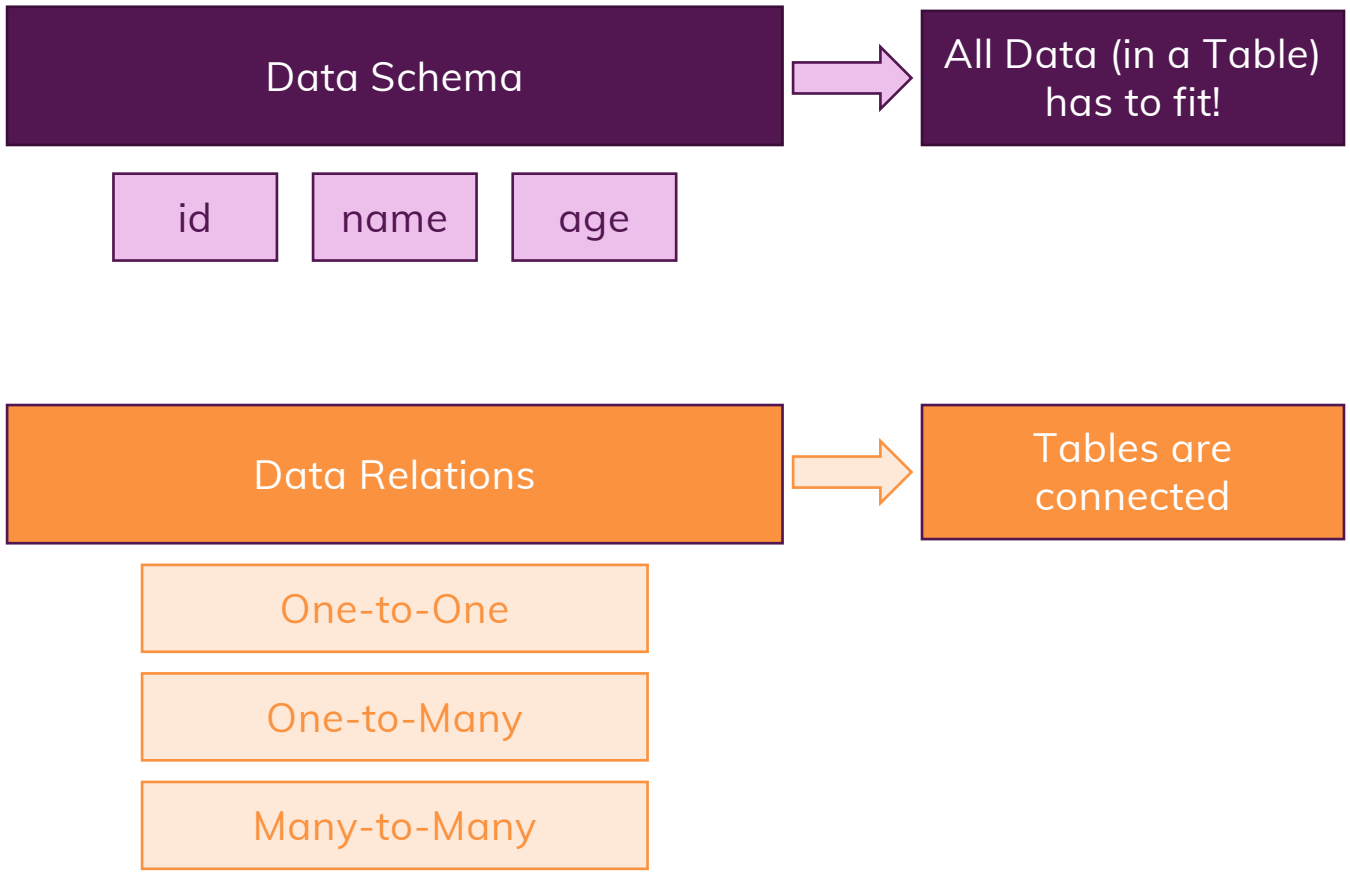

Goal: Store Data and Make it Easily Accessible
Use a Database!
Quicker Access than with a File
SQL Databases
NoSQL Databases
e.g. MySQL
e.g. MongoDB

ACADEMIND

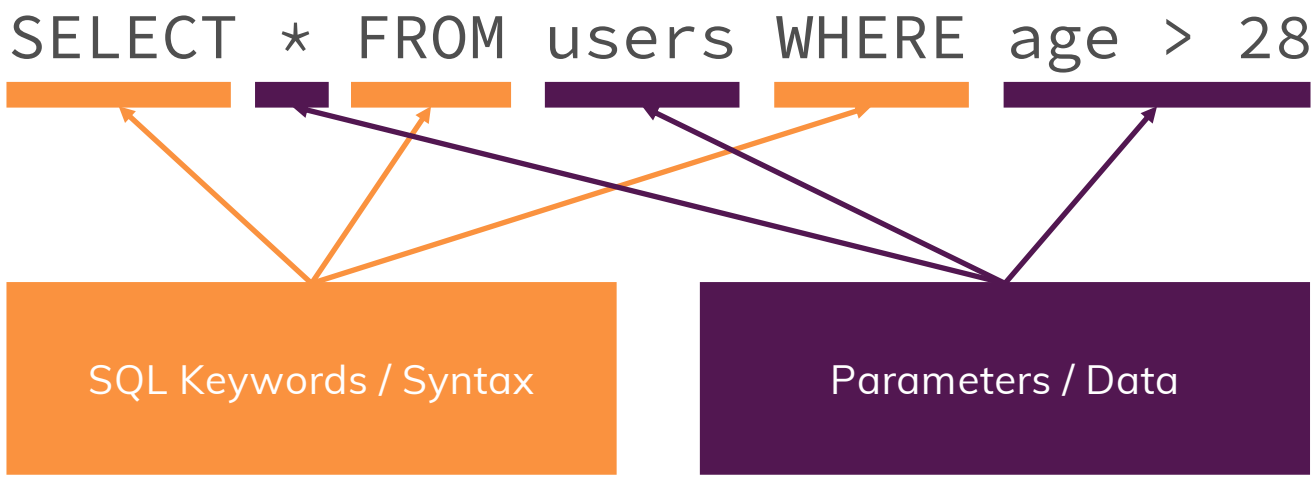
What's SQL?



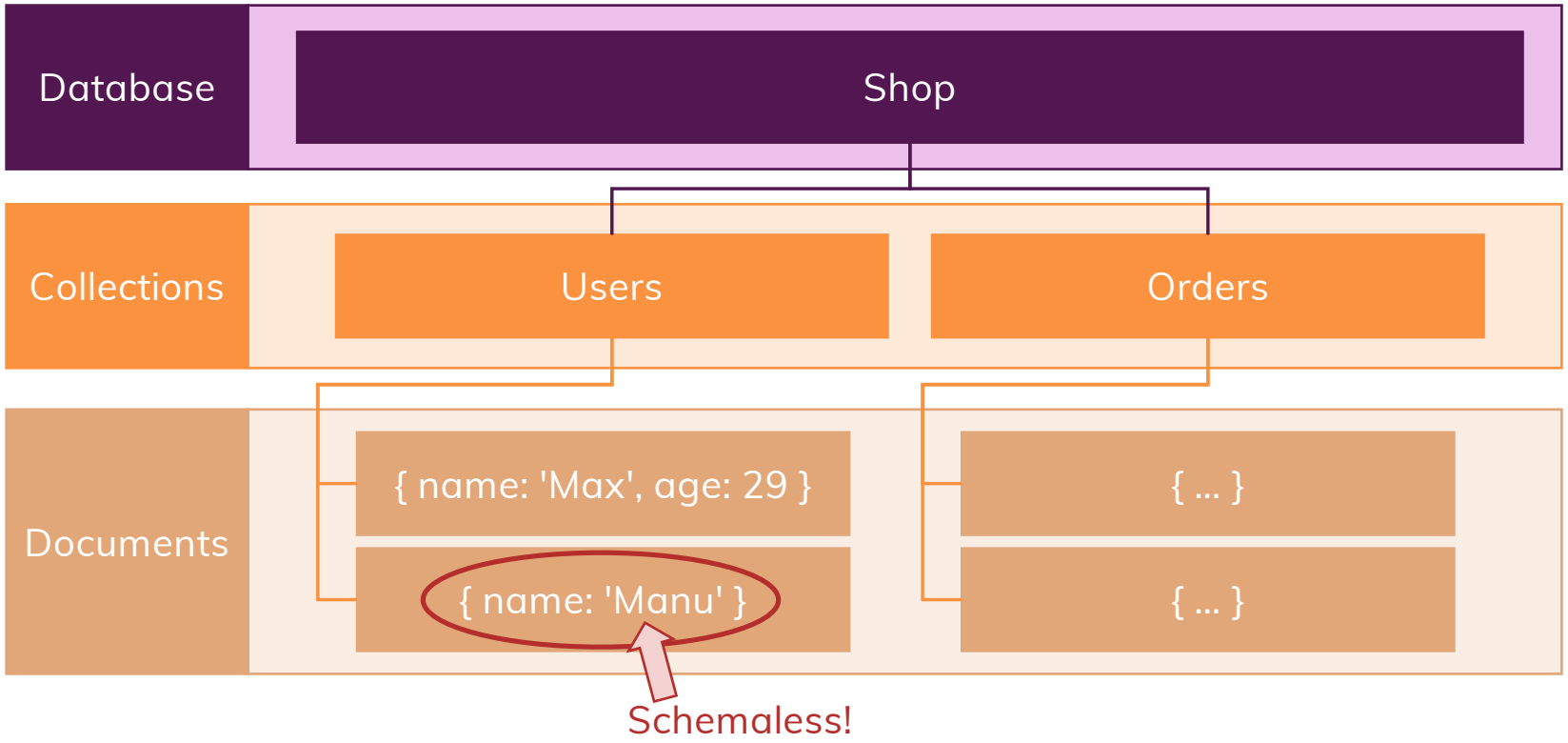
Core SQL Database Characteristics



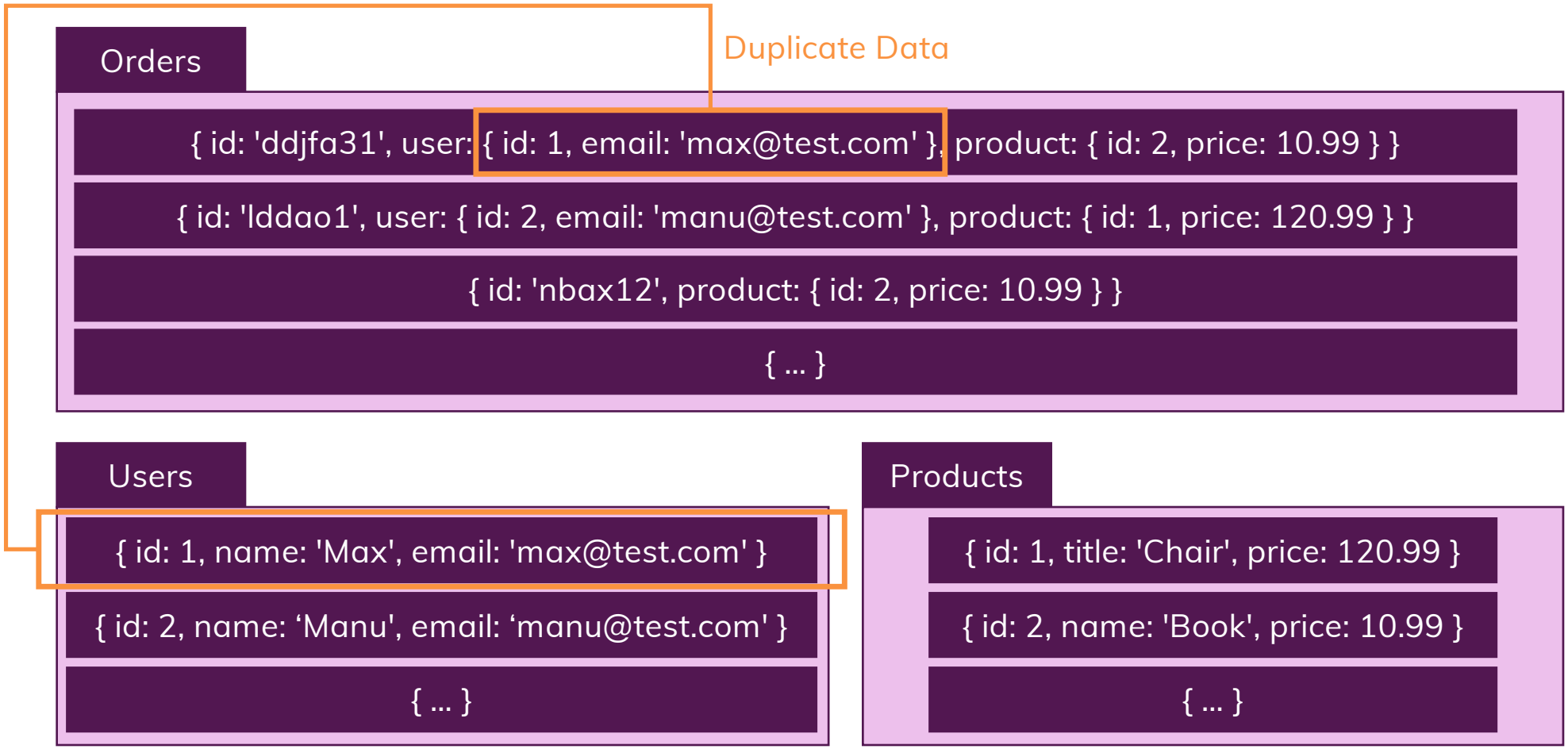
SQL Queries


SELECT * FROM users WHERE age > 28
SQL Keywords / Syntax
Parameters / Data

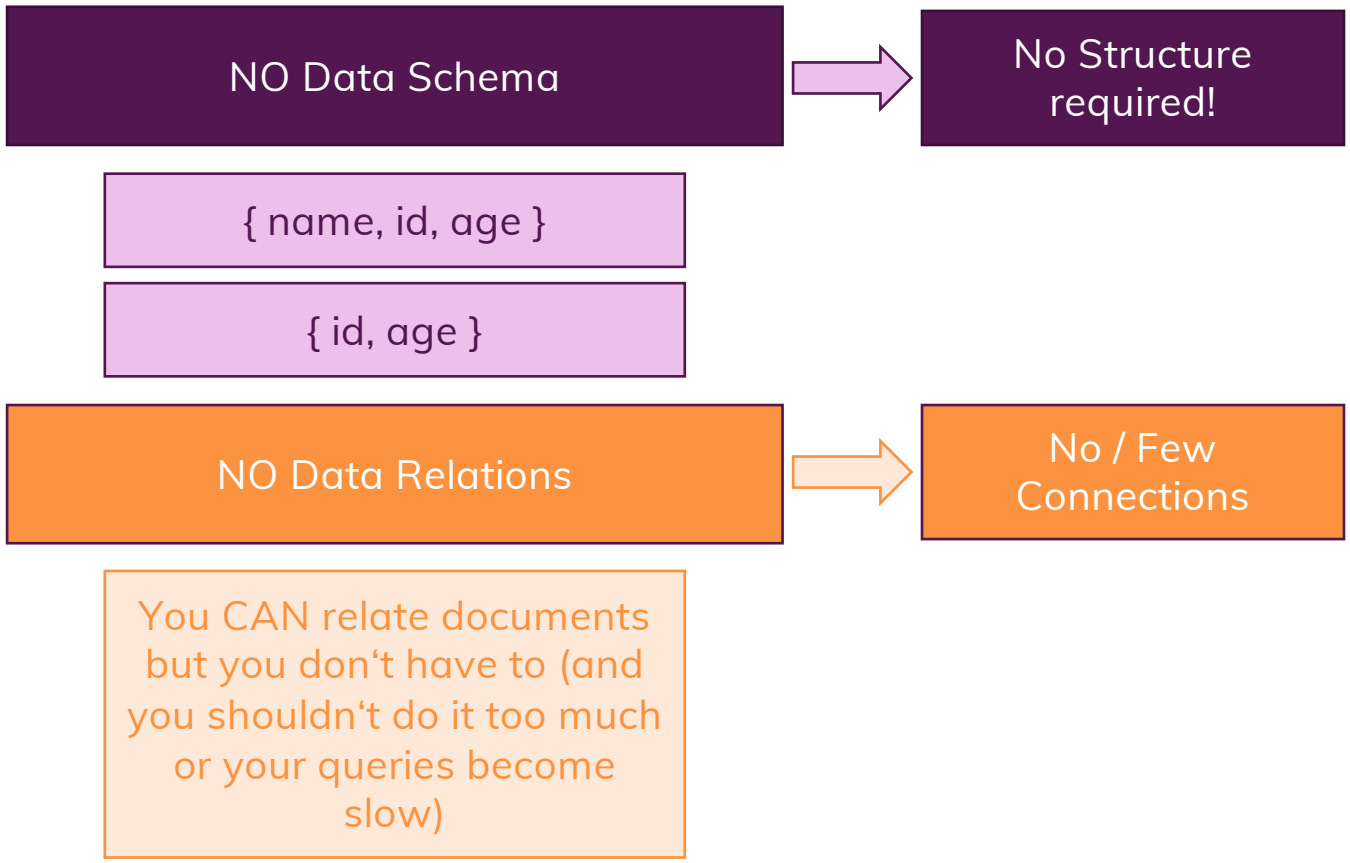
NoSQL


Database
Shop
Collections
Users
Orders
Documents
{ name: 'Max', age: 29 }
{ ... }
{ name: 'Manu' }
{ ... }
Schemaless!

What's NoSQL?

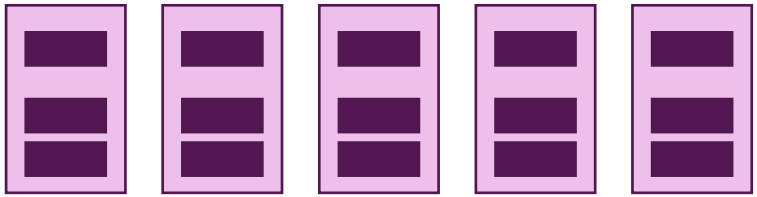

Orders
Duplicate Data
{ id: 'ddjfa31', user: { id: 1, email: 'max@test.com' }, product: { id: 2, price: 10.99 } }
{ id: 'lddao1', user: { id: 2, email: 'manu@test.com' }, product: { id: 1, price: 120.99 } }
{ id: 'nbax12', product: { id: 2, price: 10.99 } }
{ ... }
Users
{ id: 1, name: 'Max', email: 'max@test.com' }
{ id: 2, name: 'Manu', email: 'manu@test.com' }
{ ... }
Products
{ id: 1, title: 'Chair', price: 120.99 }
{ id: 2, name: 'Book', price: 10.99 }
{ ... }

NoSQL Characteristics


NO Data Schema
No Structure required!
{ name, id, age }
{ id, age }
NO Data Relations
No / Few Connections
You CAN relate documents but you don‘t have to (and you shouldn‘t do it too much or your queries become slow)

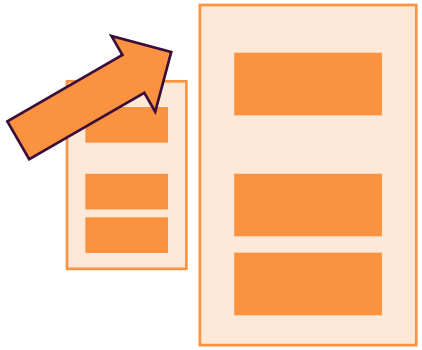
Horizontal vs Vertical Scaling

Horizontal Scaling



Add More Servers (and merge Data into one Database)

Vertical Scaling



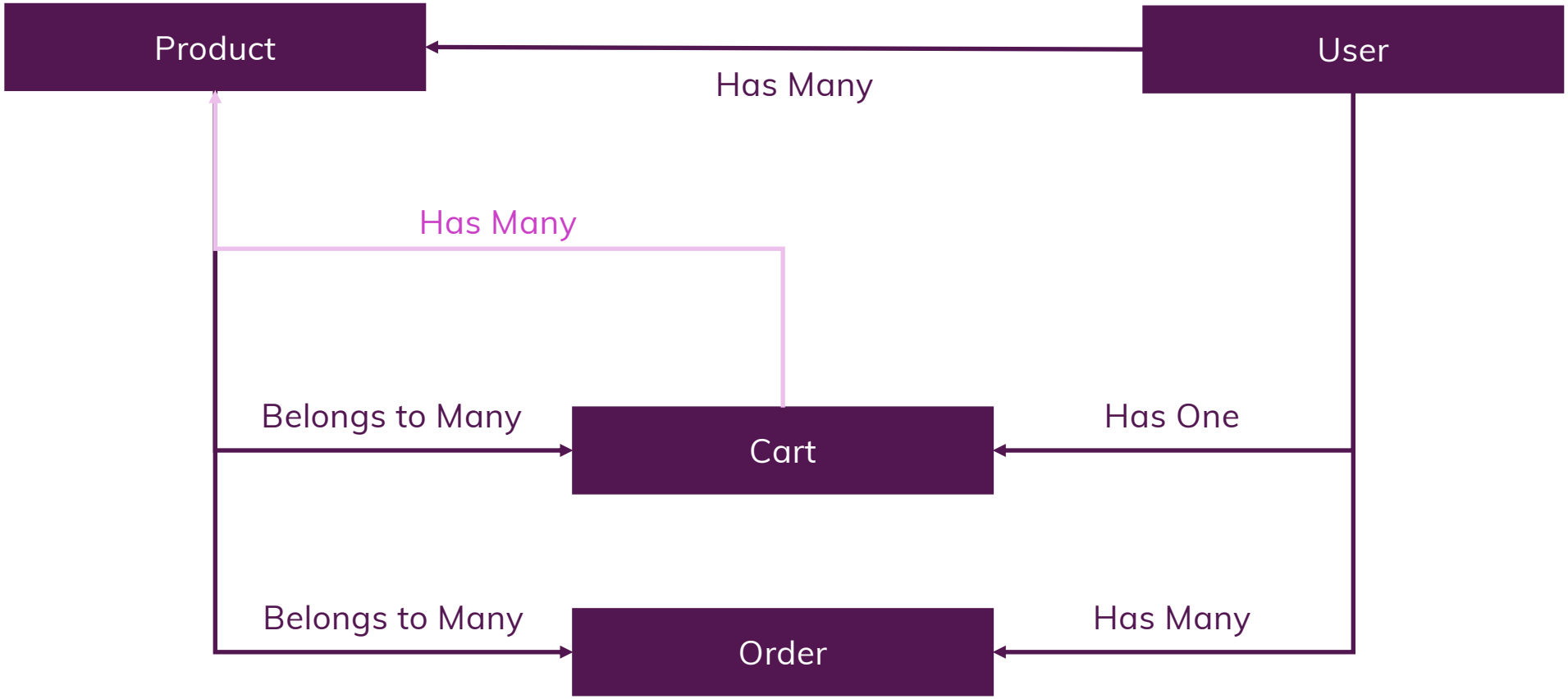
Improve Server Capacity / Hardware

ACADEMIND

SQL vs NoSQL

SQL	NoSQL
Data uses Schemas	Schema-less
Relations!	No (or very few) Relations
Data is distributed across multiple tables	Data is typically merged / nested in a few collections
Horizontal scaling is difficult / impossible; Vertical scaling is possible	Both horizontal and vertical scaling is possible
Limitations for lots of (thousands) read & write queries per second	Great performance for mass read & write requests

Associations



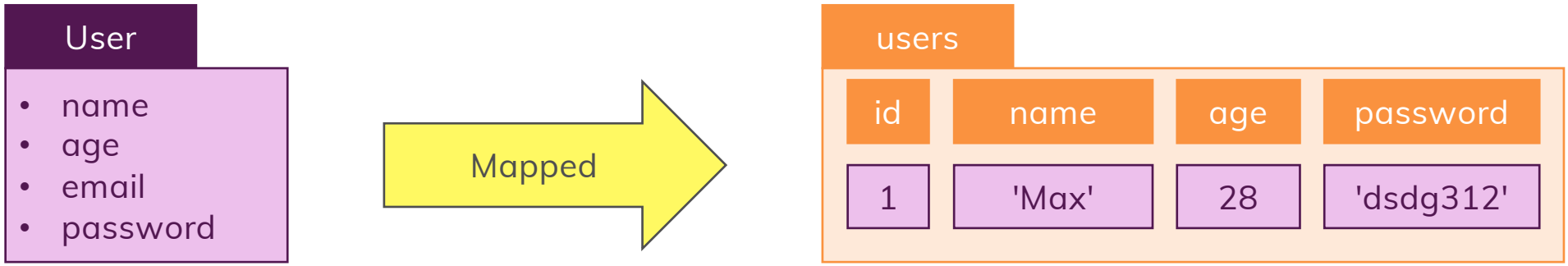
ACADEMIND

Sequelize

Focus on Node.js, not on SQL

What is Sequelize?

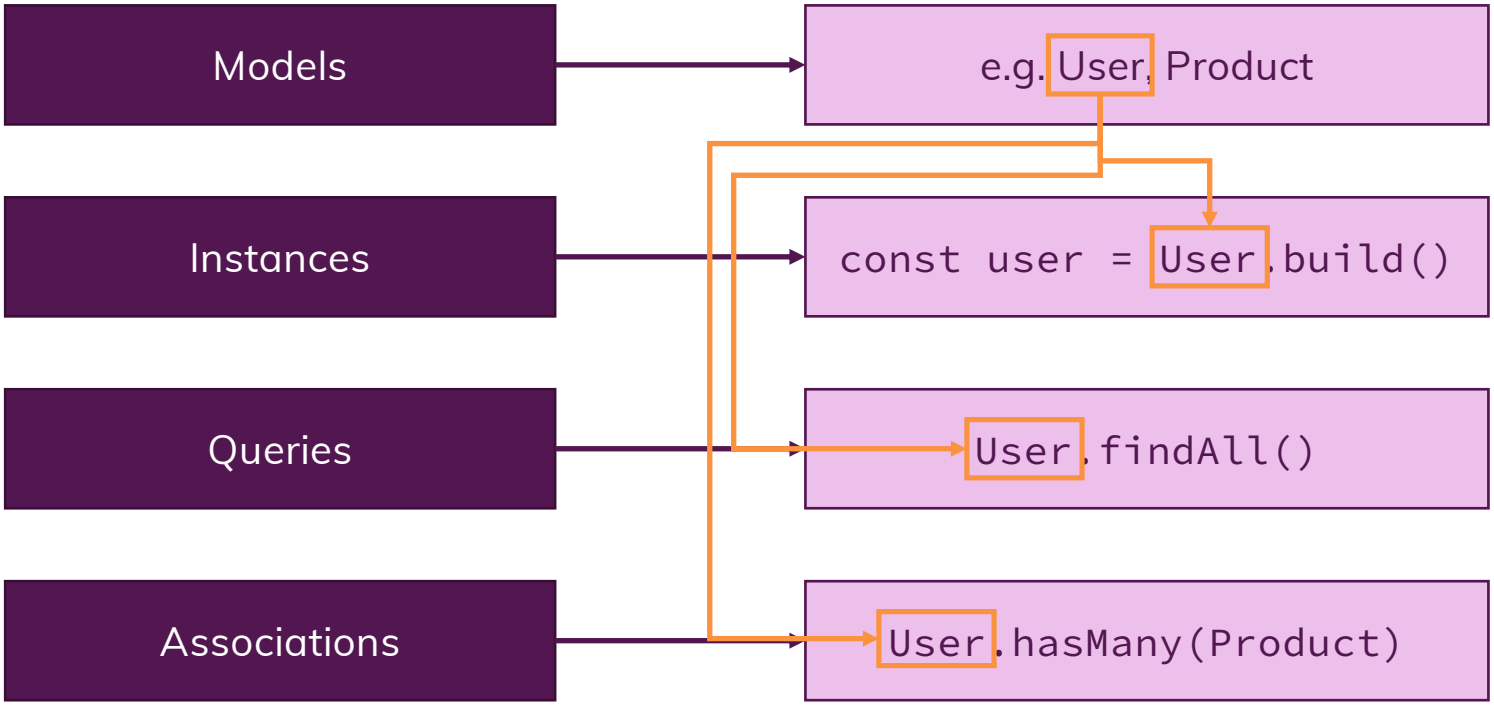
An Object-Relational Mapping Library




```
INSERT INTO users VALUES (1, 'Max', 28, 'dsdg312')
```

```
const user = User.create({ name: 'Max', age: 28, password: 'dsdg312' })
```

Core Concepts



ACADEMIND

Module Summary

SQL

- SQL uses strict data schemas and relations
- You can connect your Node.js app via packages like mysql2
- Writing SQL queries is not directly related to Node.js and something you have to learn in addition to Node.js

Sequelize

- Instead of writing SQL queries manually, you can use packages (ORMs) like Sequelize to focus on the Node.js code and work with native JS objects
- Sequelize allows you define models and interact with the database through them
- You can also easily set up relations ("Associations") and interact with your related models through them

ACADEMIND

NoSQL Databases / MongoDB

Storing Data in a Different Kind of Database

ACADEMIND

What's In This Module?

- What is MongoDB?
- Using the MongoDB Driver in Node.js Apps

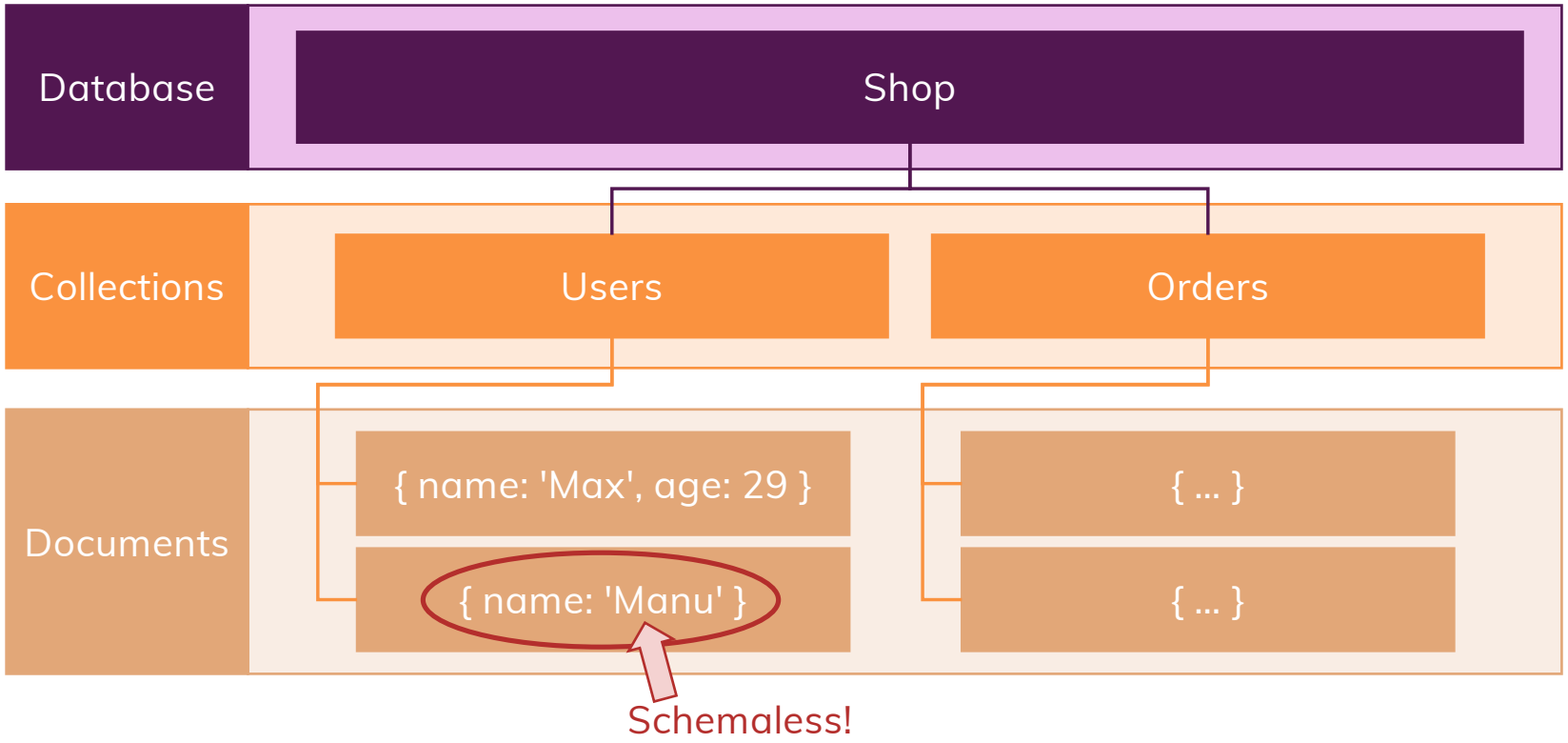
ACADEMIND

What?

Humongous

Because it can store lots and lots of data

How it works

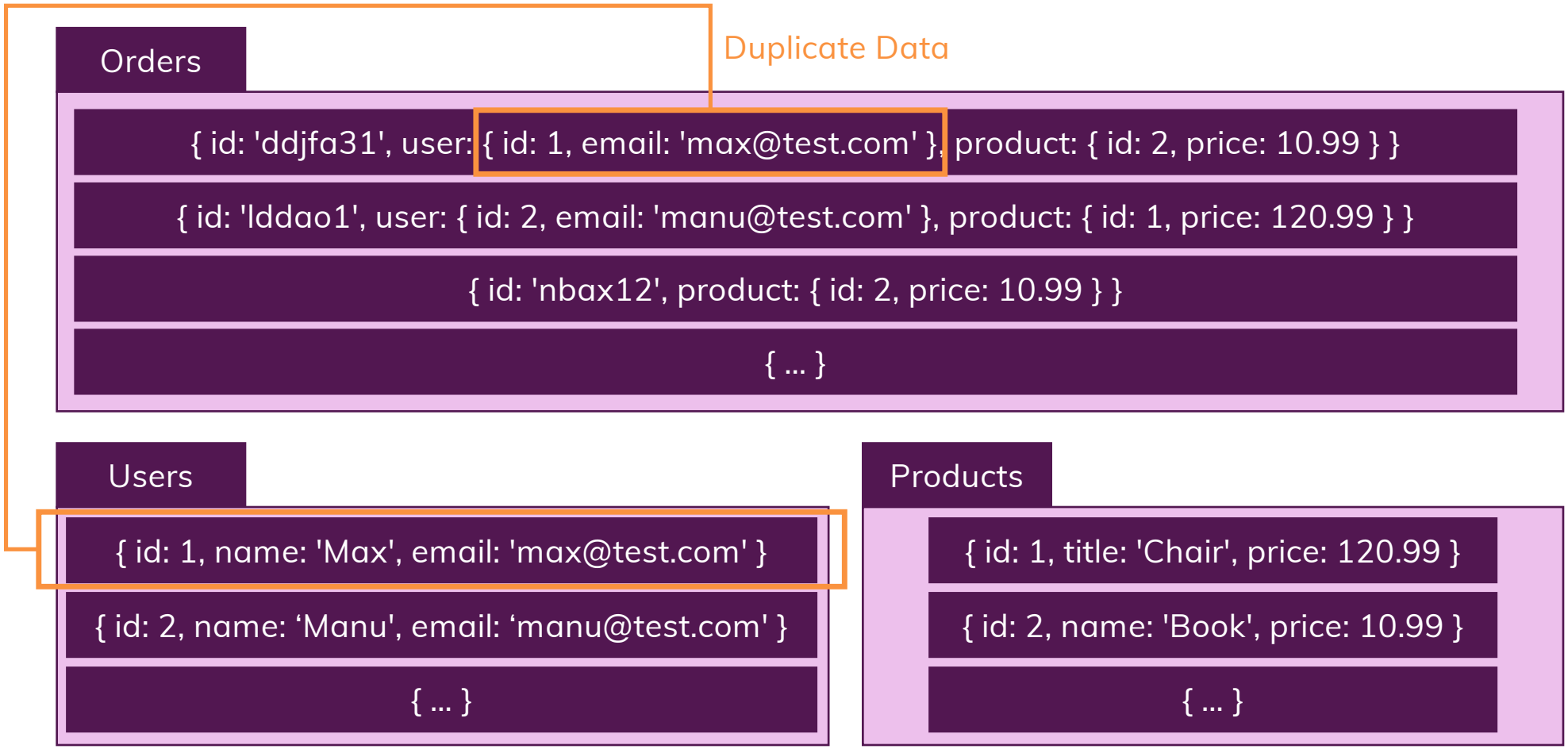

Database
Shop
Collections
Users
Orders
Documents
{ name: 'Max', age: 29 }
{ name: 'Manu' }
{ ... }
{ ... }
Schemaless!

ACADEMIND

JSON (BSON) Data Format

```
{
      "name": "Max",
      "age": 29,
      "address":
            {
                  "city": "Munich"
            },
      "hobbies": [
            { "name": "Cooking" },
            { "name": "Sports" }
      ]
}
```

What's NoSQL?


Orders
Duplicate Data
{ id: 'ddjfa31', user: { id: 1, email: 'max@test.com' }, product: { id: 2, price: 10.99 } }
{ id: 'lddao1', user: { id: 2, email: 'manu@test.com' }, product: { id: 1, price: 120.99 } }
{ id: 'nbax12', product: { id: 2, price: 10.99 } }
{ ... }
Users
{ id: 1, name: 'Max', email: 'max@test.com' }
{ id: 2, name: 'Manu', email: 'manu@test.com' }
{ ... }
Products
{ id: 1, title: 'Chair', price: 120.99 }
{ id: 2, name: 'Book', price: 10.99 }
{ ... }

Relations - Options

Nested / Embedded Documents

Customers

```
{
  userName: 'max',
  age: 29,
  address: {
    street: 'Second Street',
    city: 'New York'
  }
}
```

References

Lots of data duplication!

```
{
  favBooks: [{…}, {…}]
}
```

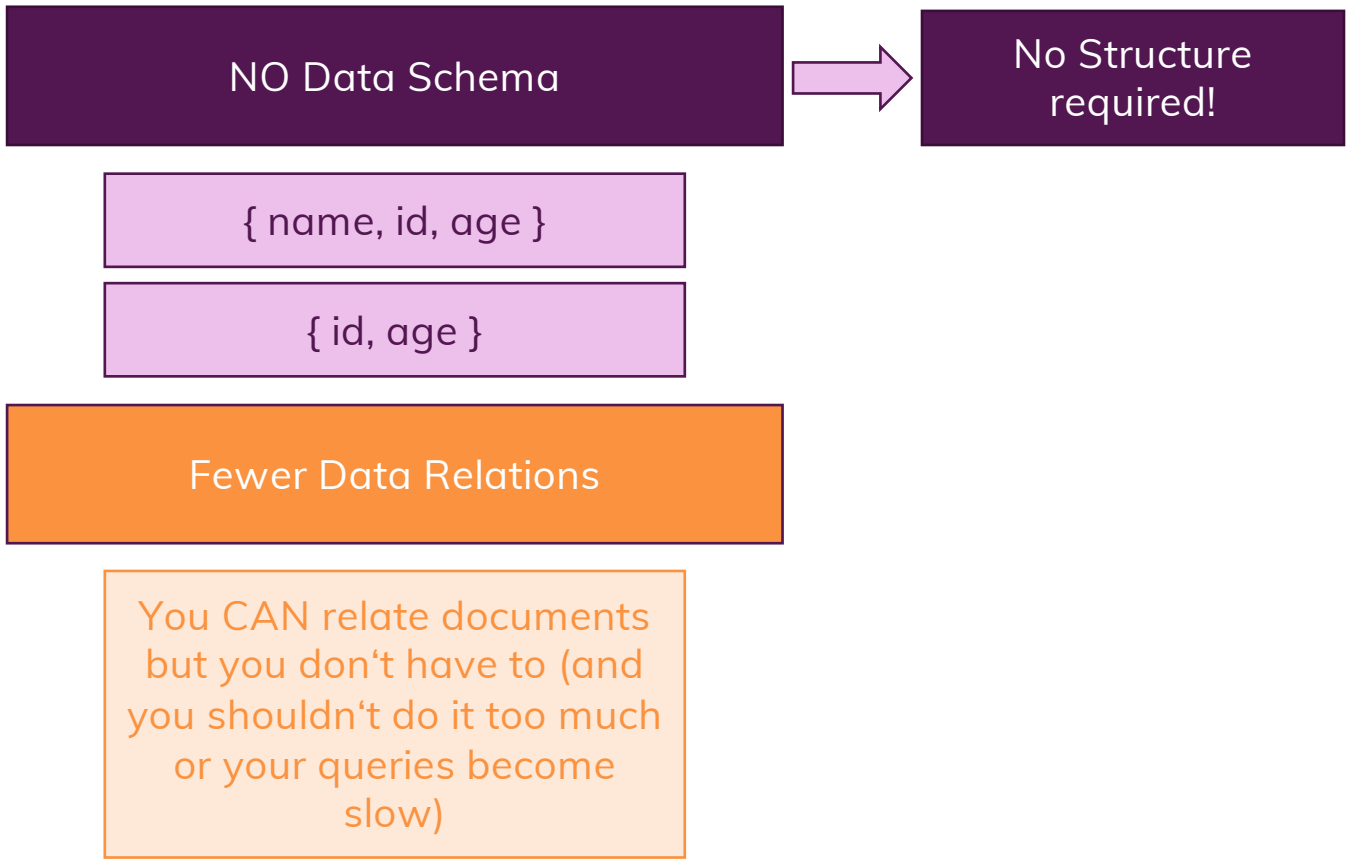
Customers

```
{
  userName: 'max',
  favBooks: ['id1', 'id2']
}
```

Books

```
{
  _id: 'id1',
  name: 'Lord of the Rings 1'
}
```

NoSQL Characteristics


NO Data Schema
No Structure required!
{ name, id, age }
{ id, age }
Fewer Data Relations
You CAN relate documents but you don‘t have to (and you shouldn‘t do it too much or your queries become slow)

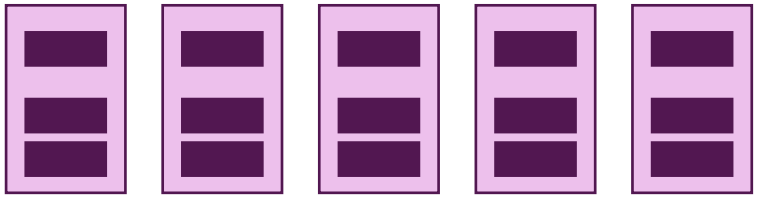
ACADEMIND

SQL vs NoSQL

SQL	NoSQL
Data uses Schemas	Schema-less
Relations!	No (or very few) Relations
Data is distributed across multiple tables	Data is typically merged / nested in a few collections
Horizontal scaling is difficult / impossible; Vertical scaling is possible	Both horizontal and vertical scaling is possible
Limitations for lots of (thousands) read & write queries per second	Great performance for mass (simple) read & write requests

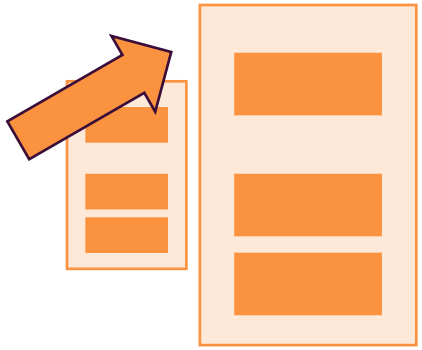
Horizontal vs Vertical Scaling

Horizontal Scaling



Add More Servers (and merge Data into one Database)

Vertical Scaling



Improve Server Capacity / Hardware

ACADEMIND

Module Summary

NoSQL / MongoDB

- Alternative to SQL databases
- No strict schemas, fewer relations
- You can of course use schemas and reference-based relations but you got more flexibility
- Often, relations are also created by embedding other documents/ data

Working with MongoDB

- Use the official MongoDB Driver
- Commands like insertOne(), find(), updateOne() and deleteOne() make CRUD-operations very simple
- Check the official docs to learn about all available operations + configurations/ operators
- All operations are promise-based, hence you can easily chain them for more complex flows

ACADEMIND

Mongoose

A MongoDB ORM

ACADEMIND

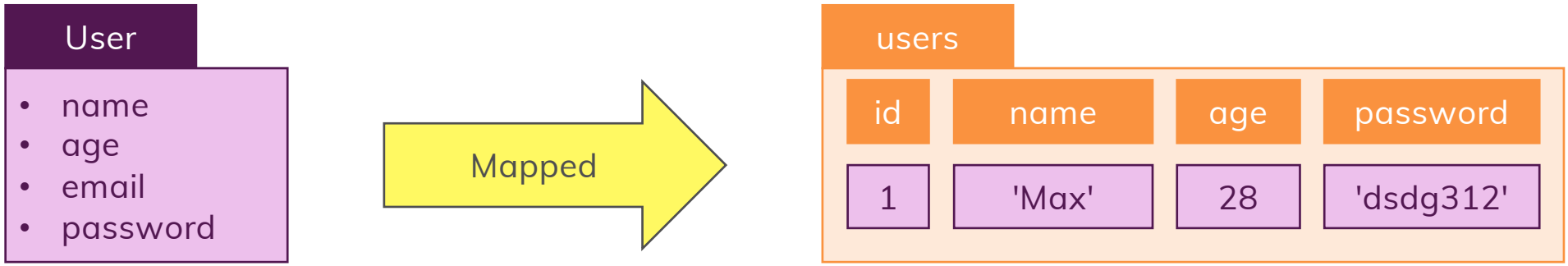
What's In This Module?

- What is Mongoose?
- Using Mongoose in Node.js Apps

ACADEMIND

What is Mongoose?

A Object-Document Mapping Library

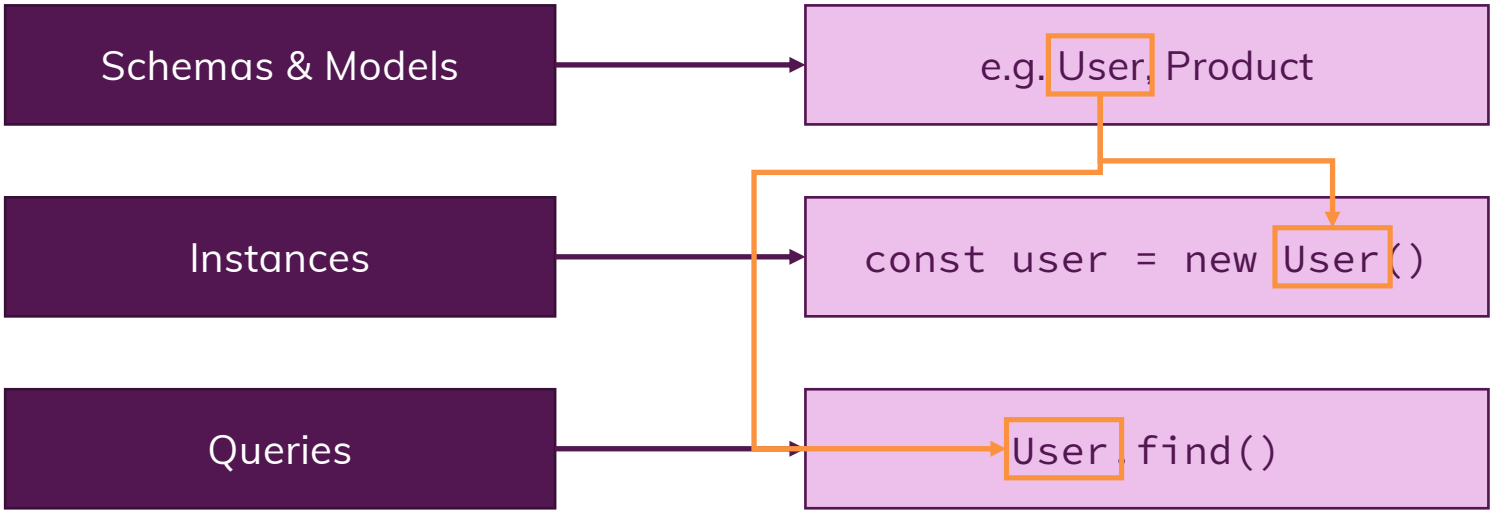



```
db.collection('users').insertO    name: 'Max', age: 28, password:
                                  'ds      ' })
```

```
const user = User.create({ name: 'Max', age: 28, password: 'dsdg312' })
```

ACADEMIND

Core Concepts



ACADEMIND

Module Summary

Mongoose

- SQL uses strict data schemas and relations
- You can connect your Node.js app via packages like mysql2
- Writing SQL queries is not directly related to Node.js and something you have to learn in addition to Node.js

Sequelize

- Instead of writing SQL queries manually, you can use packages (ORMs) like Sequelize to focus on the Node.js code and work with native JS objects
- Sequelize allows you define models and interact with the database through them
- You can also easily set up relations ("Associations") and interact with your related models through them

ACADEMIND

Sessions & Cookies

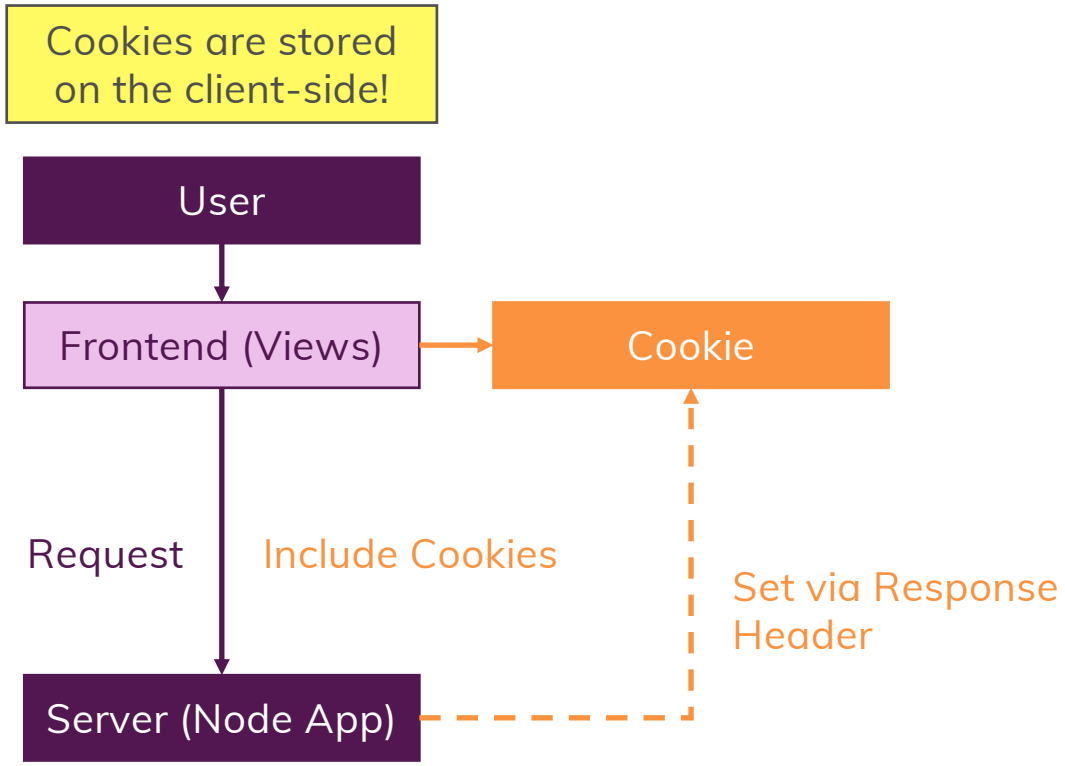
Persisting Data across Requests

ACADEMIND

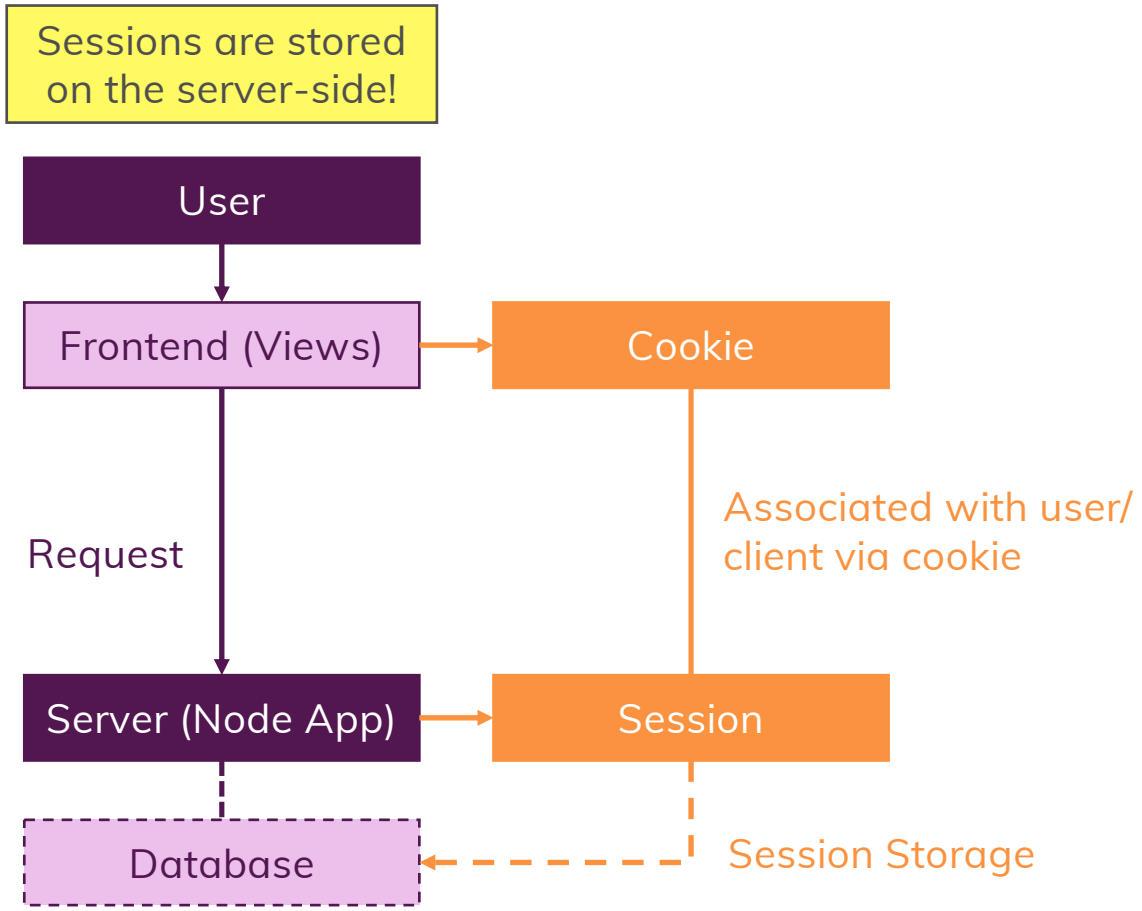
What's In This Module?

- What are Cookies?
- What are Sessions?
- Using Session & Cookies?

What's a Cookie?



What's a Session?



When to use What

Cookies	Session
Stored on client	Stored on server
(Ad) Tracking	Authentication Status Management (across Requests)
Authentication Session Management	General Cross-Request Data Management

ACADEMIND

Module Summary

Cookies

- Great for storing data on the client (browser)
- Do NOT store sensitive data here! It can be viewed + manipulated
- Cookies can be configured to expire when the browser is closed (=> “Session Cookie”) or when a certain age/ expiry date is reached (“Permanent Cookie”)
- Works well together with Sessions...

Sessions

- Stored on the server, NOT on the client
- Great for storing sensitive data that should survive across requests
- You can store ANYTHING in sessions
- Often used for storing user data/ authentication status
- Identified via Cookie (don’t mistake this with the term “Session Cookie)
- You can use different storages for saving your sessions on the server

ACADEMIND

User Authentication

Restricting Access

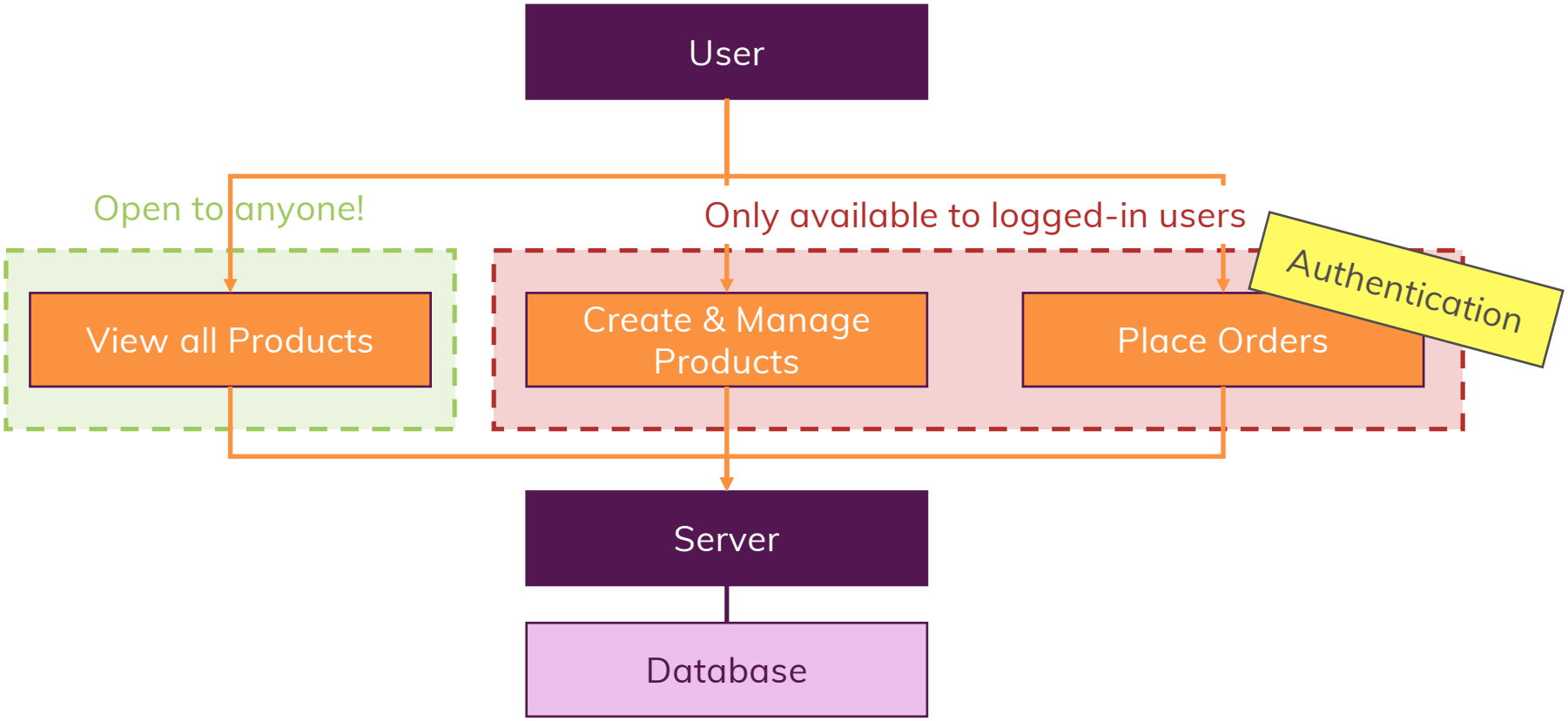
ACADEMIND

What's In This Module?

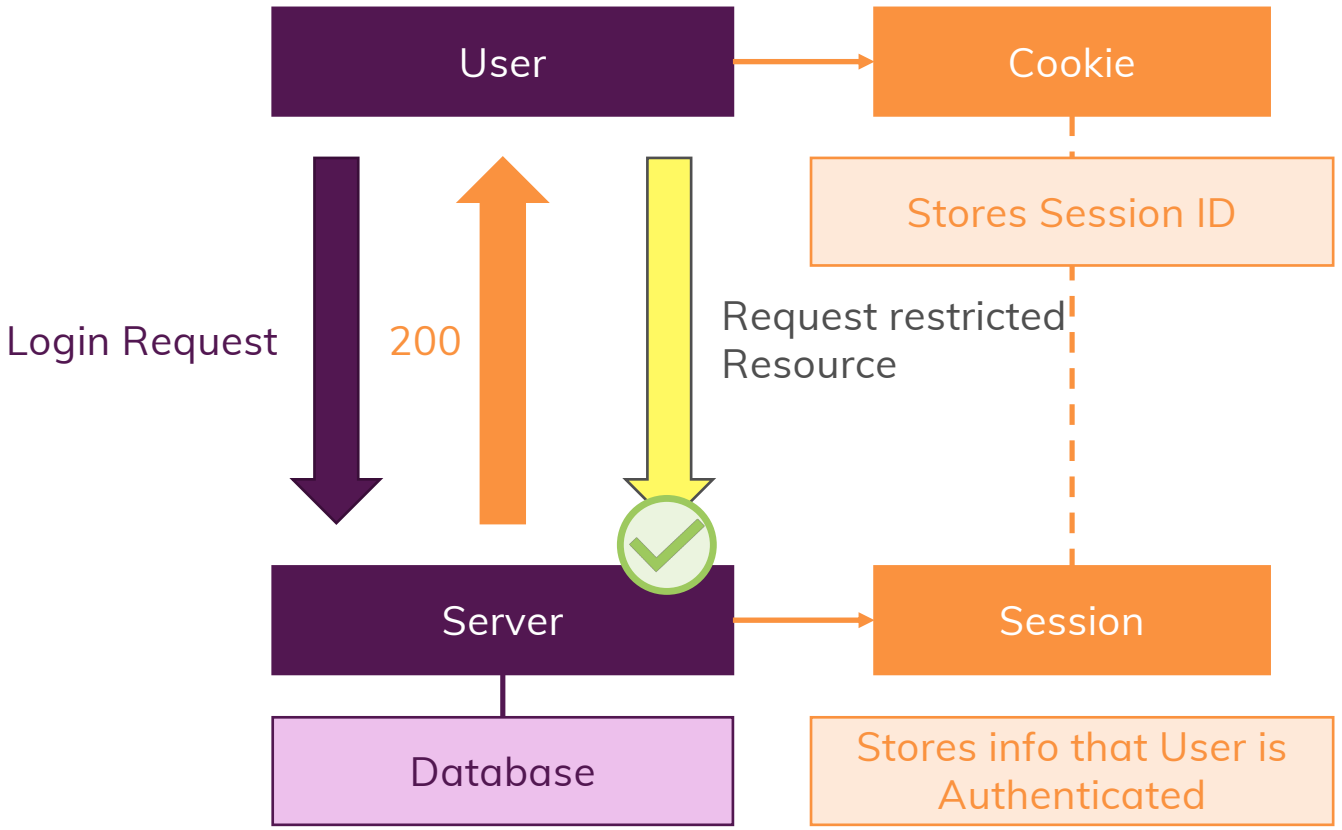
- What exactly is "Authentication"?
- Storing & Using Credentials
- Protecting Routes

ACADEMIND

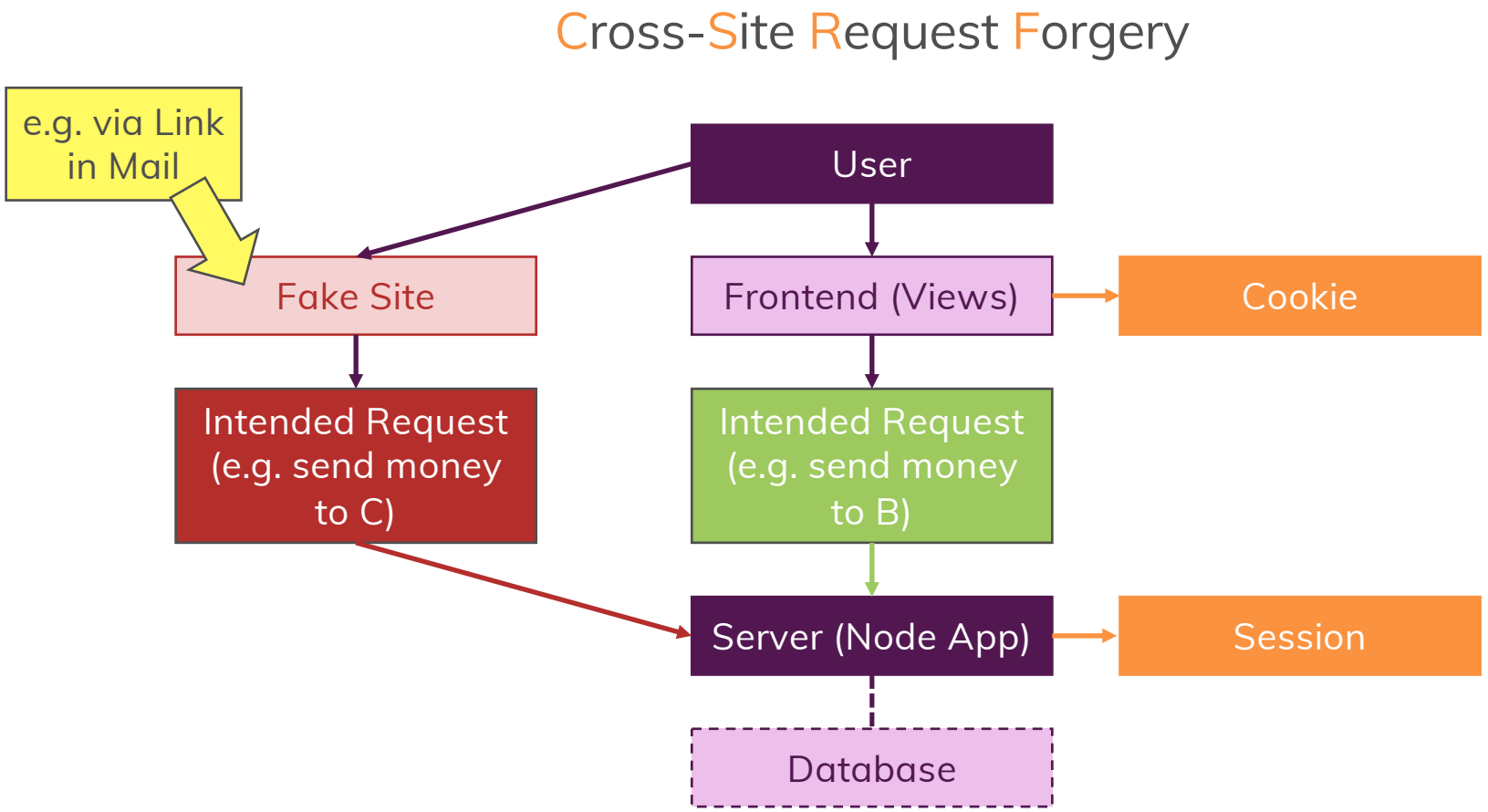
What is “Authentication”?



How is Authentication Implemented?



CSRF Attacks



Module Summary

Authentication

- Authentication means that not every visitor of the page can view and interact with everything
- Authentication has to happen on the server-side and builds up on sessions
- You can protect routes by checking the (session-controlled) login status right before you access a controller action

Security & UX

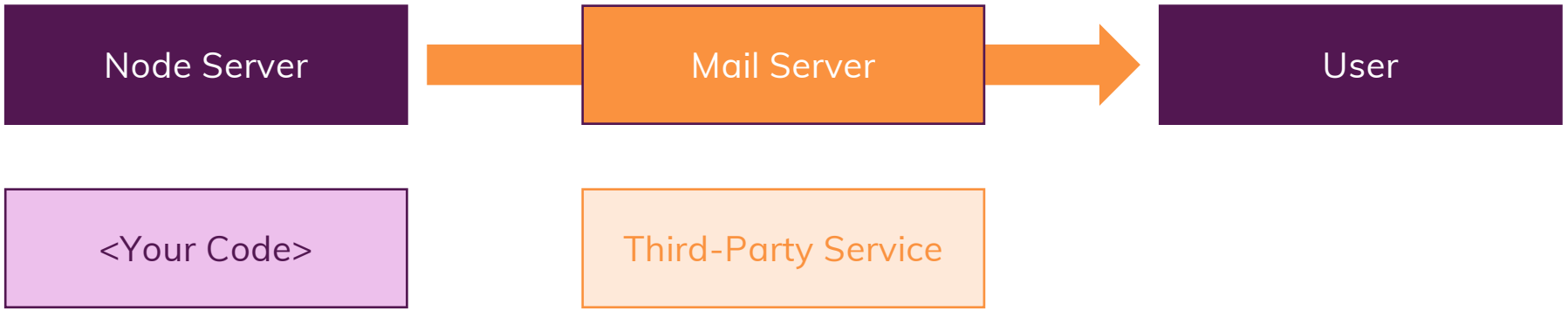
- Passwords should be stored in a hashed form
- CSRF attacks are a real issue and you should therefore include CSRF protection in ANY application you build!
- For a better user experience, you can flash data/ messages into the session which you then can display in your views

ACADEMIND

Sending Mails

Communicating with the Outside World

How Does Sending Mails Work?



ACADEMIND

Advanced Authentication & Authorization

Beyond Signup & Login

ACADEMIND

What's In This Module?

- Resetting Passwords
- Authorization

ACADEMIND

Module Summary

Password Resetting

- Password resetting has to be implemented in a way that prevents users from resetting random user accounts
- Reset tokens have to be random, unguessable and unique

Authorization

- Authorization is an important part of pretty much every app
- Not every authenticated user should be able to do everything
- Instead, you want to lock down access by restricting the permissions of your users

ACADEMIND

Forms, User Input & Validation

Getting that Precious User Input

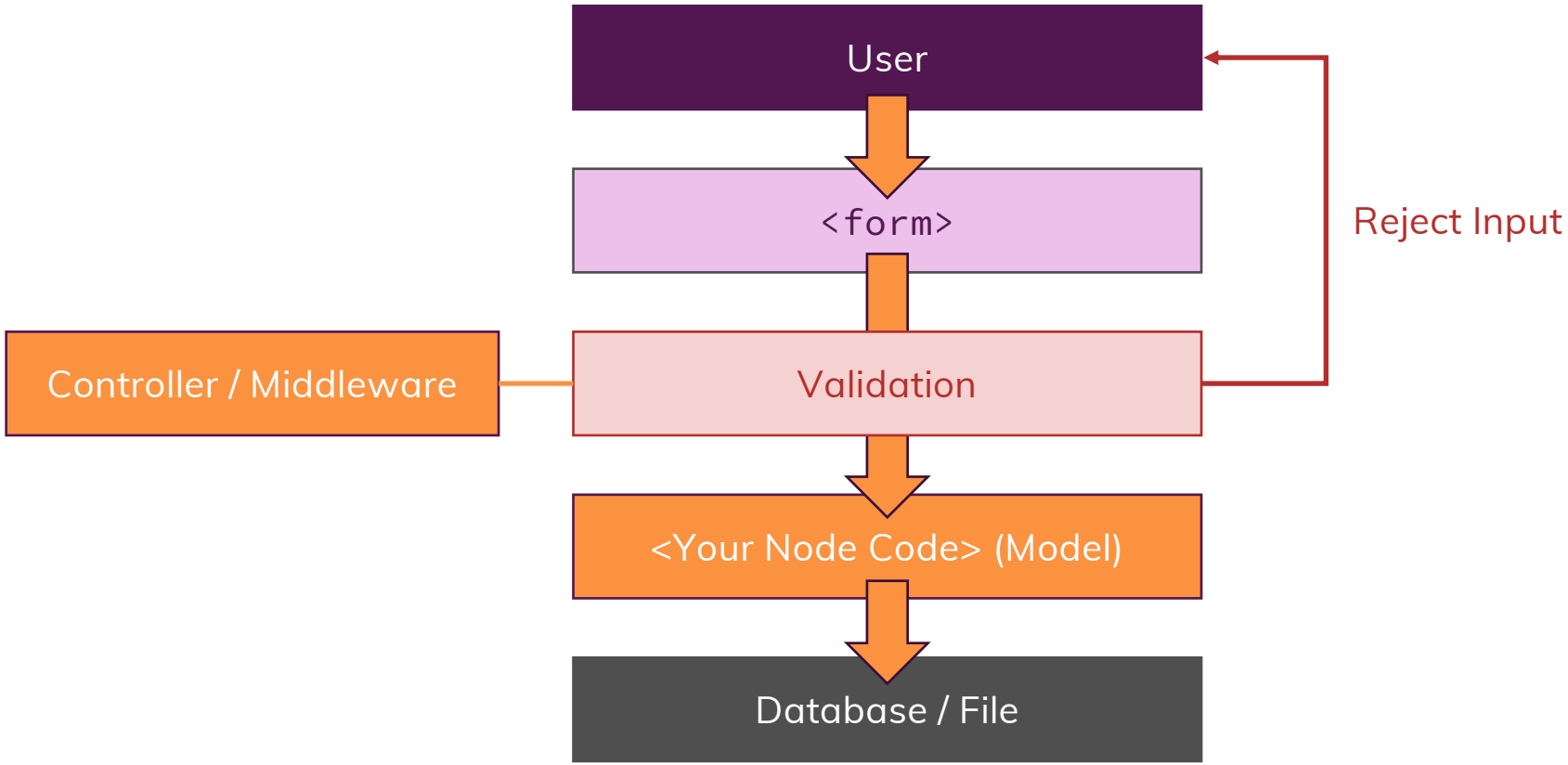
ACADEMIND

What's In This Module?

- Why Validate?
- How to Validate

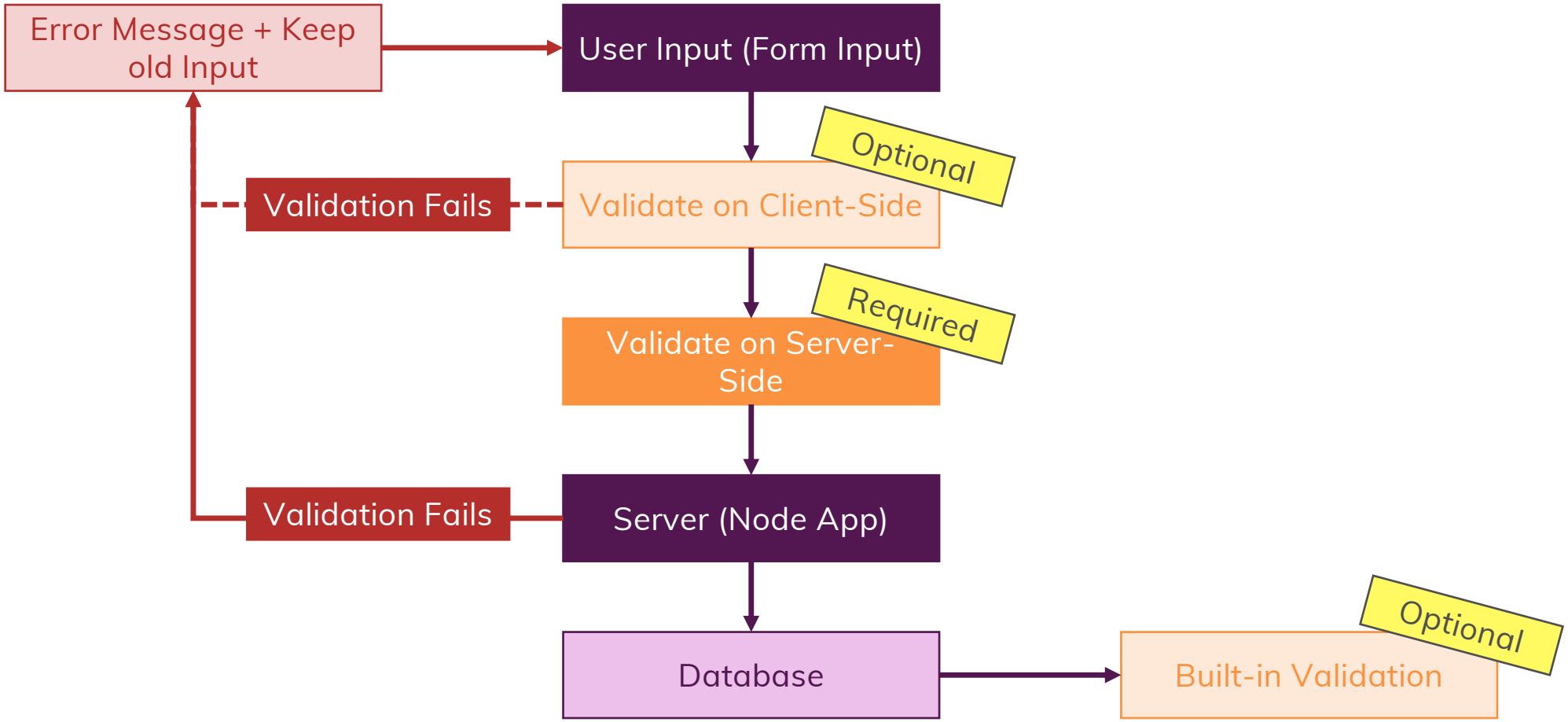
ACADEMIND

Why Validate?



ACADEMIND

How to Validate



ACADEMIND

Error Handling

Fail Gracefully

ACADEMIND

What's In This Module?

- Different Types of Errors
- Handling Errors

ACADEMIND

How Bad Are Errors?

Errors are not necessarily the end of your app!

You just need to handle them correctly!

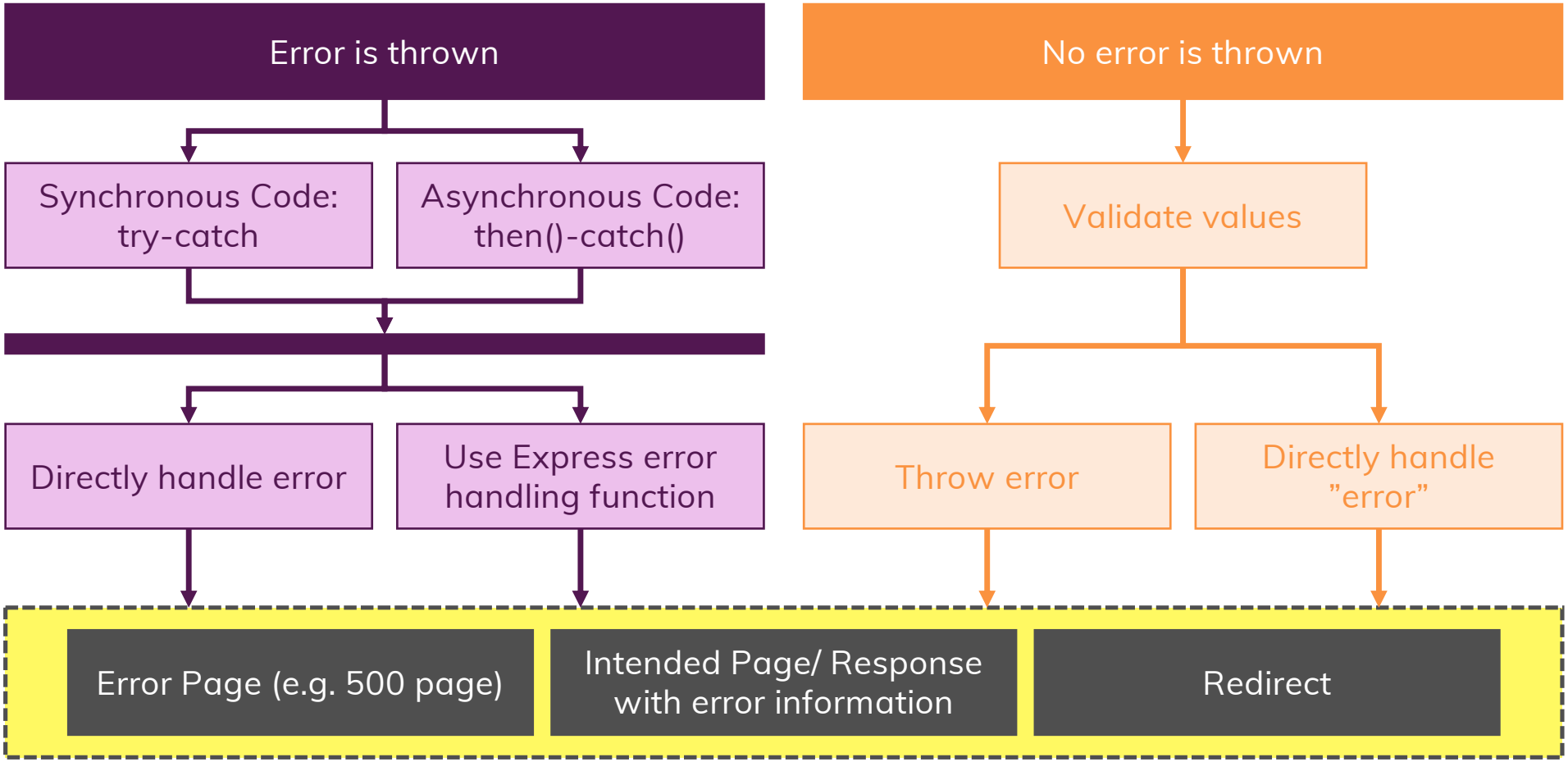
ACADEMIND

Different Types of Errors

Technical/ Network Errors	"Expected" Errors	Bugs/ Logical Errors
e.g. MongoDB server is down	e.g. File can't be read, database operation fails	e.g. User object used when it doesn't exist
Show error page to user	Inform user, possibly retry	Fix during development!

ACADEMIND

Working with Errors



ACADEMIND

Errors & Http Response Codes

2xx (Success)	200	Operation succeeded
	201	Success, resource created
3xx (Redirect)	301	Moved permanently
4xx (Client-side error)	401	Not authenticated
	403	Not authorized
	404	Not found
	422	Invalid input
5xx (Server-side error)	500	Server-side error

ACADEMIND

Module Summary

Types of Errors & Handling Errors

- You can differentiate between different types of errors – technical errors (which are thrown) and ”expected errors” (e.g. invalid user input)
- Error handling can be done with custom if-checks, try-catch, then()-catch() etc
- You can use the Express error handling middleware to handle all unhandled errors

Errors & Status Code

- When returning responses, it can make sense to also set an appropriate Http status code – this lets the browser know what went wrong
- You got success (2xx), redirect (3xx), client-side errors (4xx) and server-side errors (5xx) codes to choose from
- Setting status codes does NOT mean that the response is incomplete or the app crashed!

File Uploads & Downloads

Handling Files Correctly

ACADEMIND

What's In This Module?

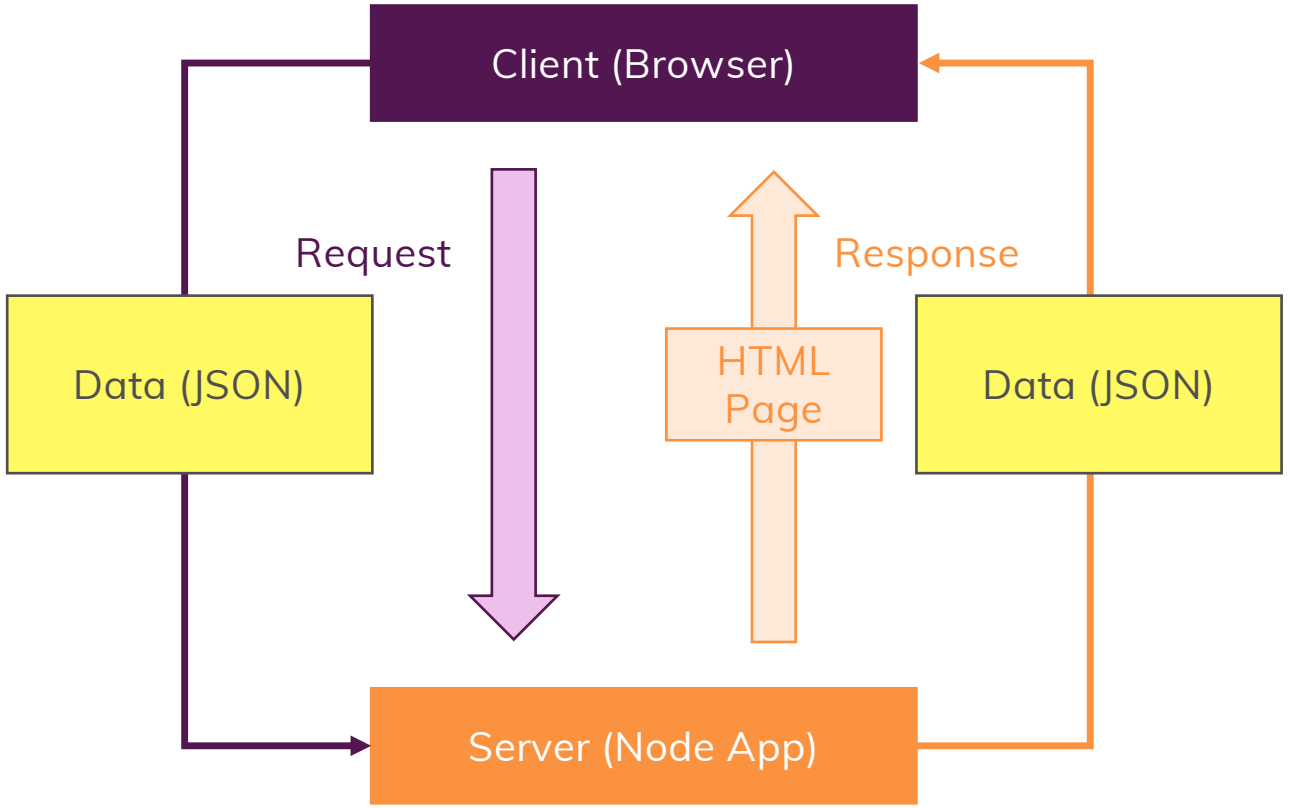
- Uploading Files
- Downloading Files

ACADEMIND

Asynchronous JavaScript Requests

Behind-The-Scenes Work

What are Asynchronous Requests?

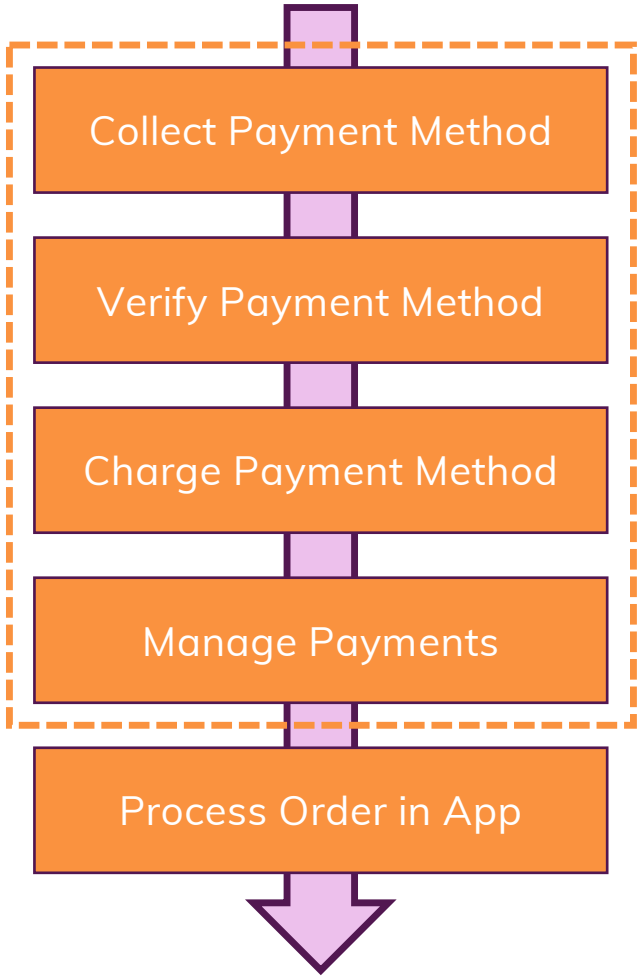


Adding Payments

Creating a Real Shop

ACADEMIND

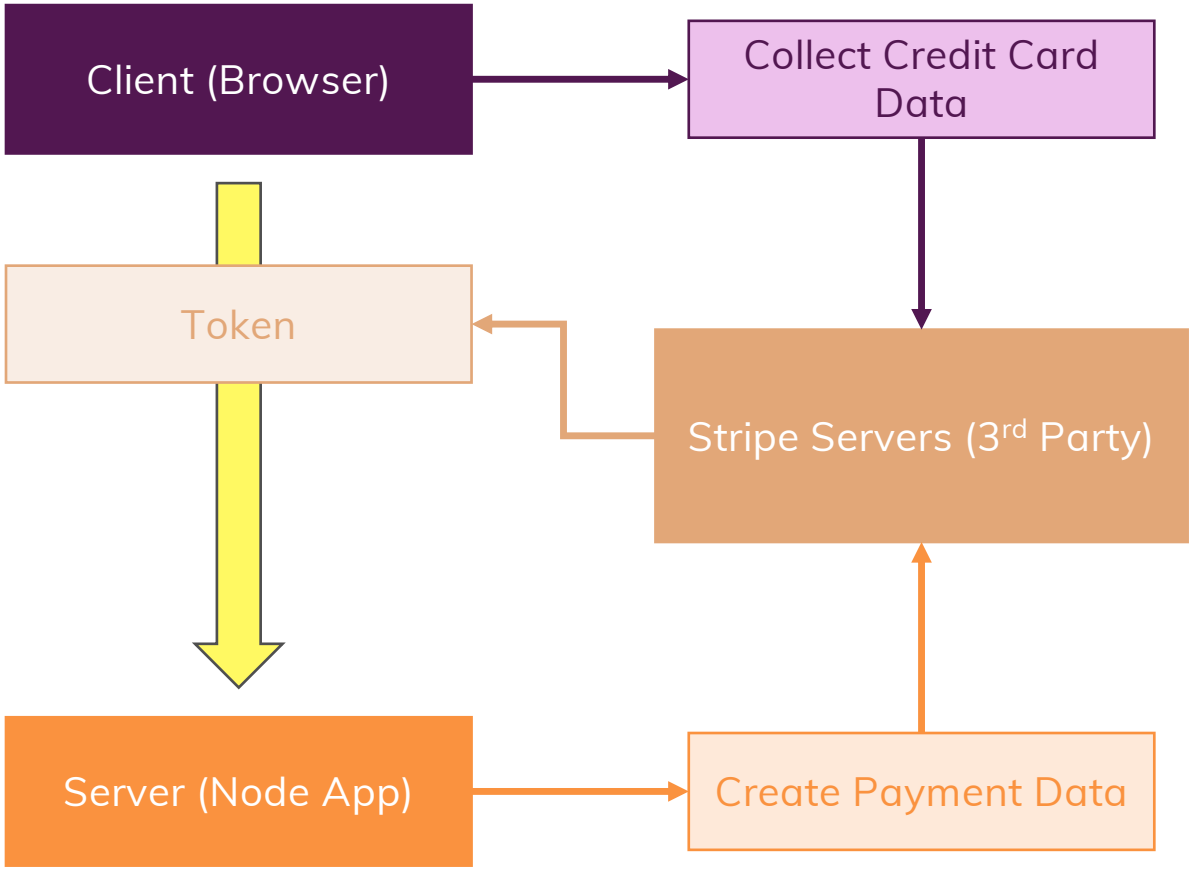
Payment Process



Complex Tasks, typically outsourced

ACADEMIND

How Stripe Works



ACADEMIND

REST APIs

Decoupling Frontend and Backend

ACADEMIND

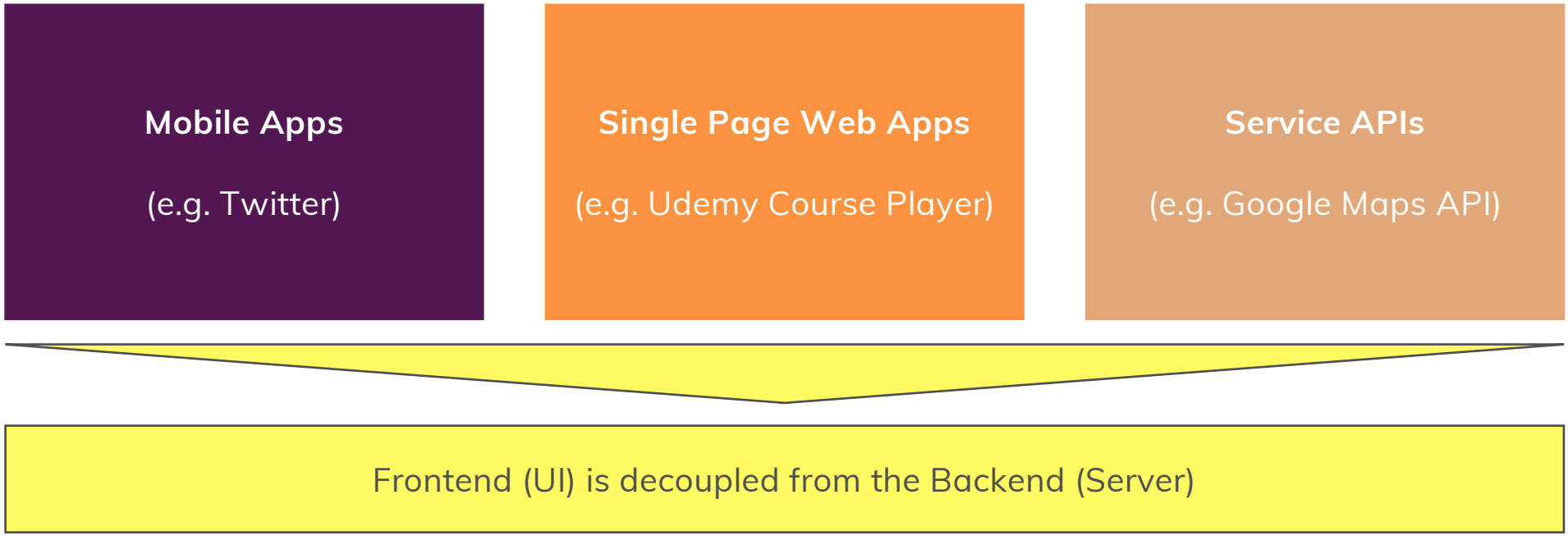
What's In This Module?

- What are "REST APIs"?
- Why use/ build REST APIs?
- Core REST Concepts & Principles
- First REST API!

ACADEMIND

What & Why?

Not every Frontend (UI) requires HTML Pages!



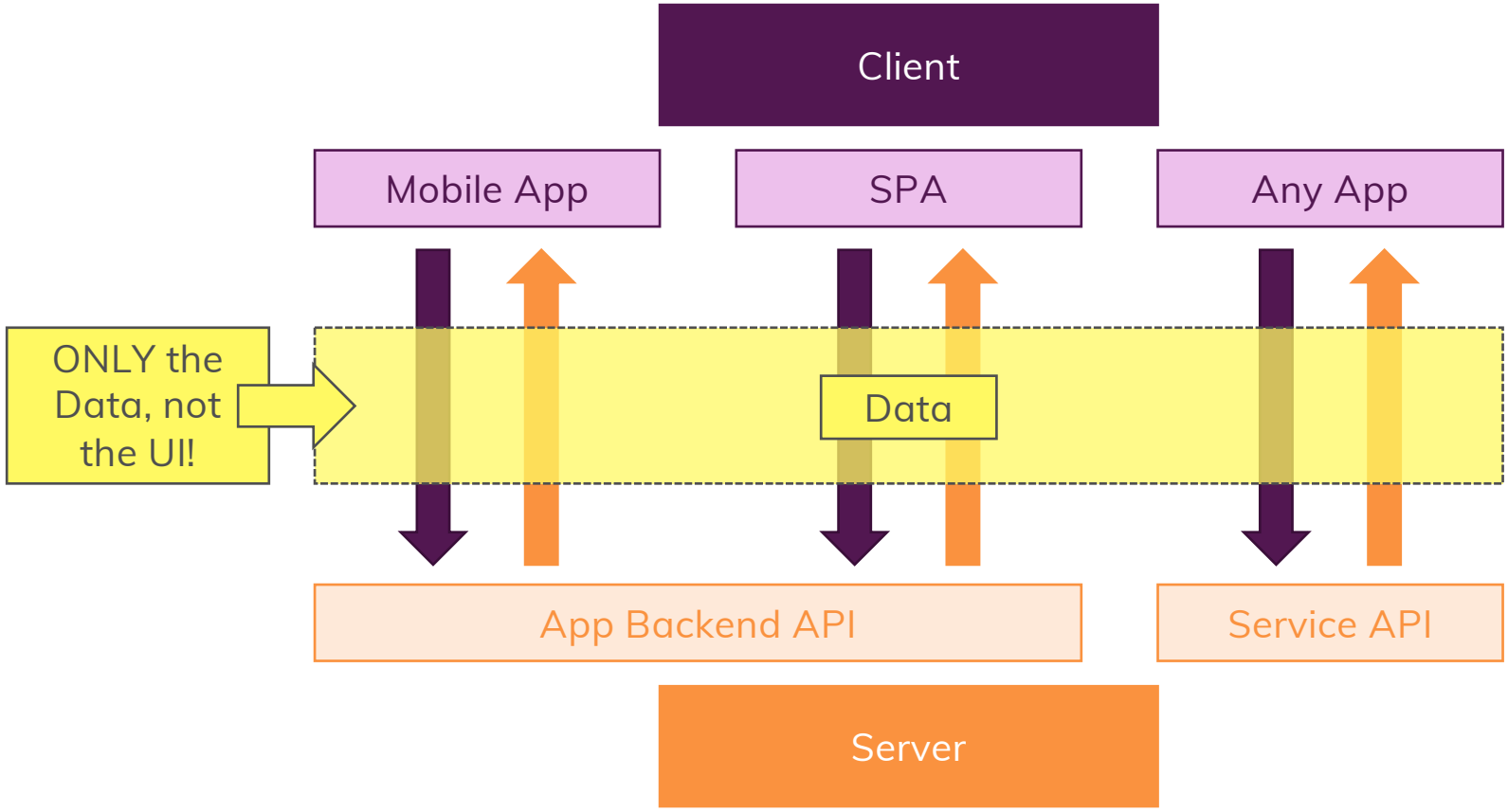
A Different Kind of Response is Needed

Representational State Transfer

Transfer Data instead of User Interfaces

Important: Only the response (and the request data) changes, NOT the general server-side logic!

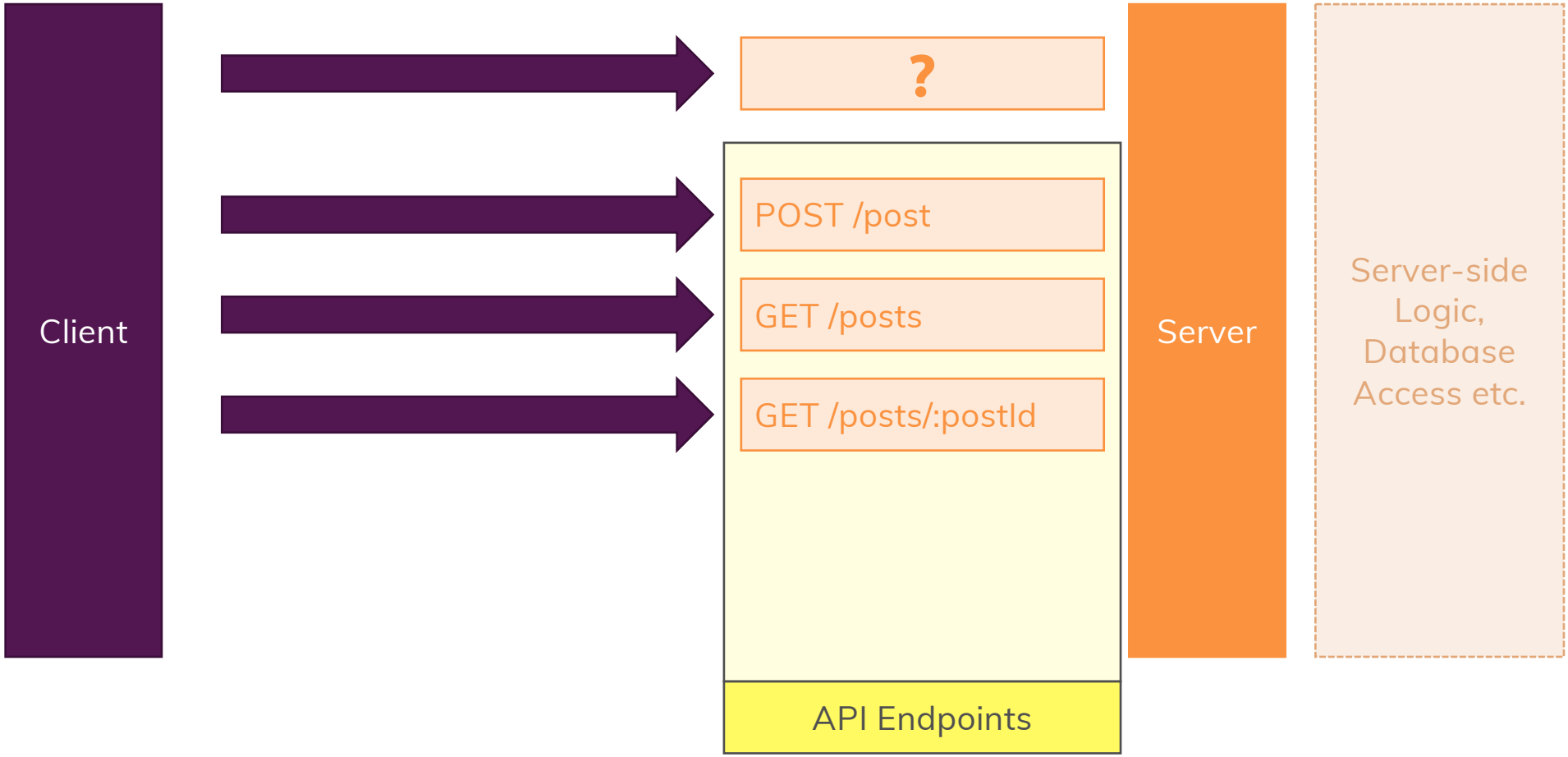
REST API Big Picture



Data Formats

HTML	Plain Text	XML	JSON
`<p>Node.js</p>`	`Node.js`	`<name>Node.js</name>`	`{"title": "Node.js"}`
Data + Structure	Data	Data	Data
Contains User Interface	No UI Assumptions	No UI Assumptions	No UI Assumptions
Unnecessarily difficult to parse if you just need the data	Unnecessarily difficult to parse, no clear data structure	Machine-readable but relatively verbose; XML-parser needed	Machine-readable and concise; Can easily be converted to JavaScript

Routing


Client
?
POST /post
GET /posts
GET /posts/:postId
API Endpoints
Server
Server-side Logic, Database Access etc.

ACADEMIND

Http Methods (Http Verbs)

More than just GET & POST

GET	POST	PUT
Get a Resource from the Server	Post a Resource to the Server (i.e. create or append Resource)	Put a Resource onto the Server (i.e. create or overwrite a Resource)

PATCH	DELETE	OPTIONS
Update parts of an existing Resource on the Server	Delete a Resource on the Server	Determine whether follow-up Request is allowed (sent automatically)

REST Principles

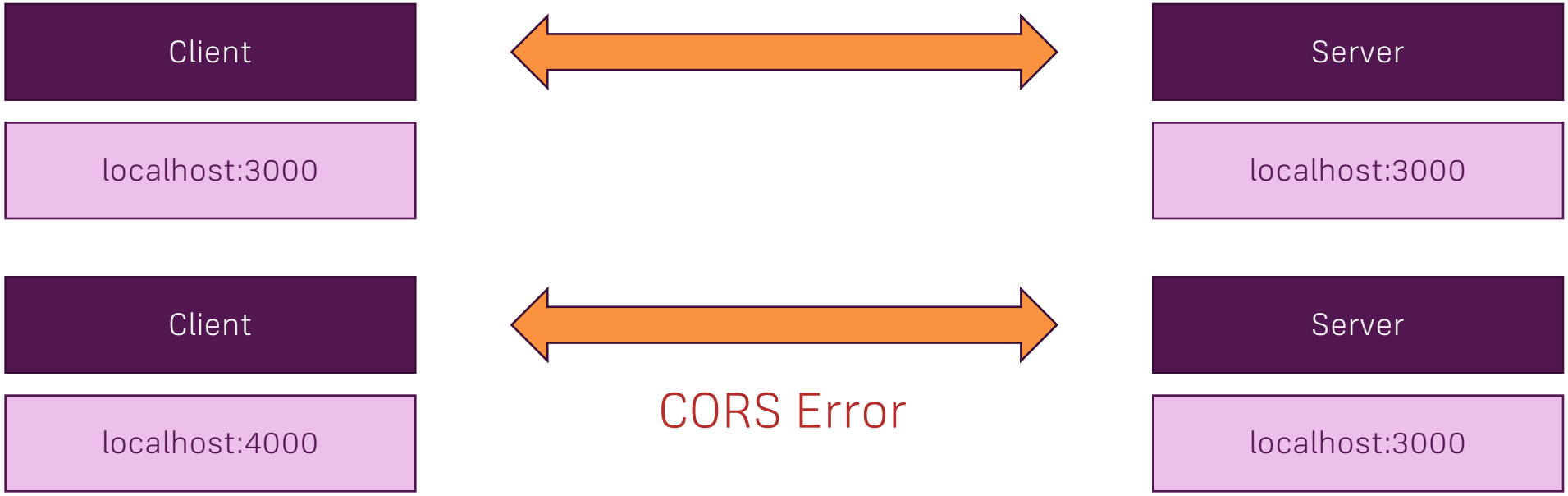
Uniform Interface	Stateless Interactions
Clearly defined API endpoints with clearly defined request + response data structure	Server and client don't store any connection history, every request is handled seperately

Cacheable	Client-Server	Layered System	Code on Demand
Servers may set caching headers to allow the client to cache responses	Server and client are separated, client is not concerned with persistent data storage	Server may forward requests to other APIs	Executable code may be transferred from server to client

ACADEMIND

CORS?

Cross-Origin Resource Sharing



Module Summary

REST Concepts & Ideas

- REST APIs are all about data, no UI logic is exchanged
- REST APIs are normal Node servers which expose different endpoints (Http method + path) for clients to send requests to
- JSON is the common data format that is used both for requests and responses
- REST APIs are decoupled from the clients that use them

Requests & Responses

- Attach data in JSON format and let the other end know by setting the “Content-Type” header
- CORS errors occur when using an API that does not set CORS headers

ACADEMIND

Advanced REST API Topics

Complete Project, Authentication & More

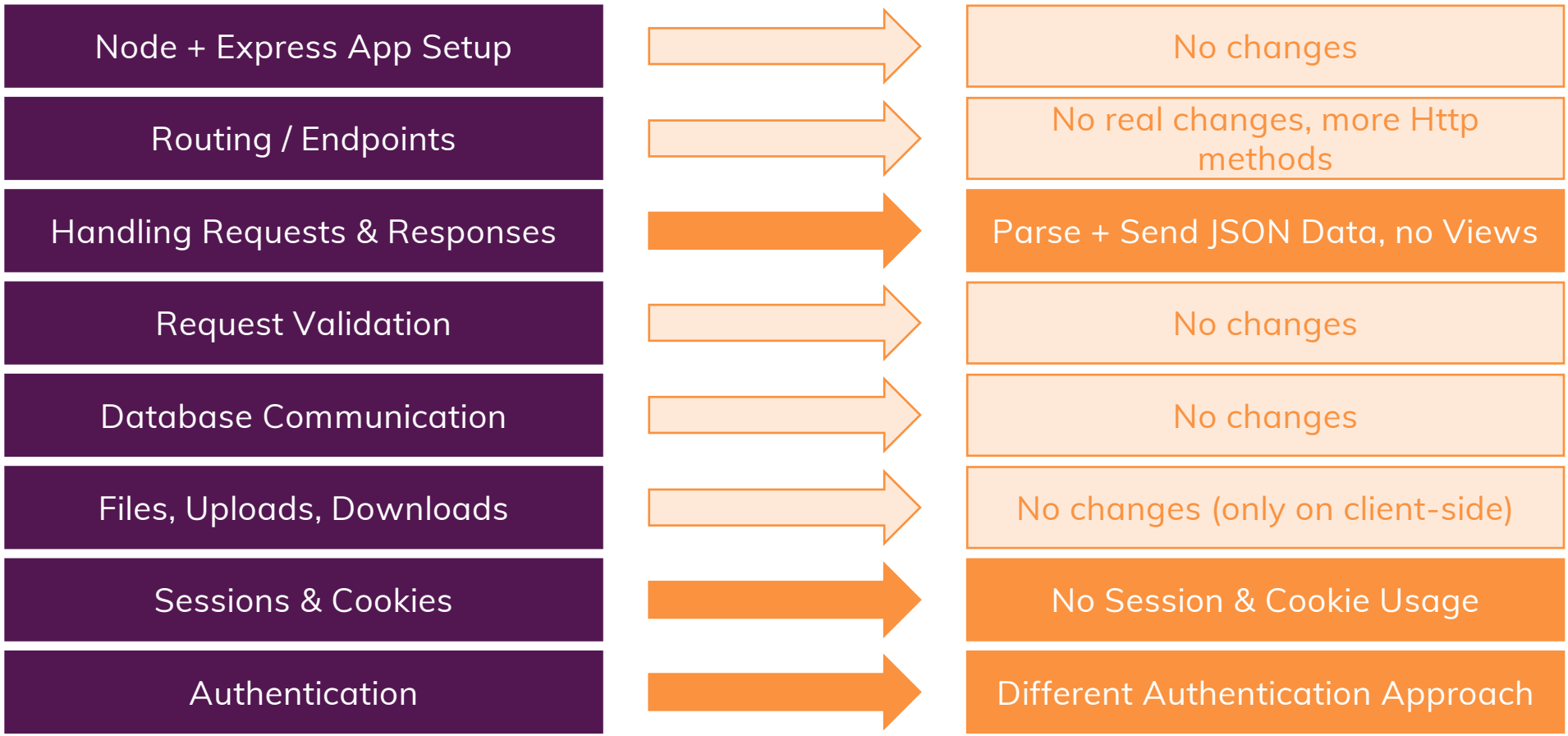
ACADEMIND

What's In This Module?

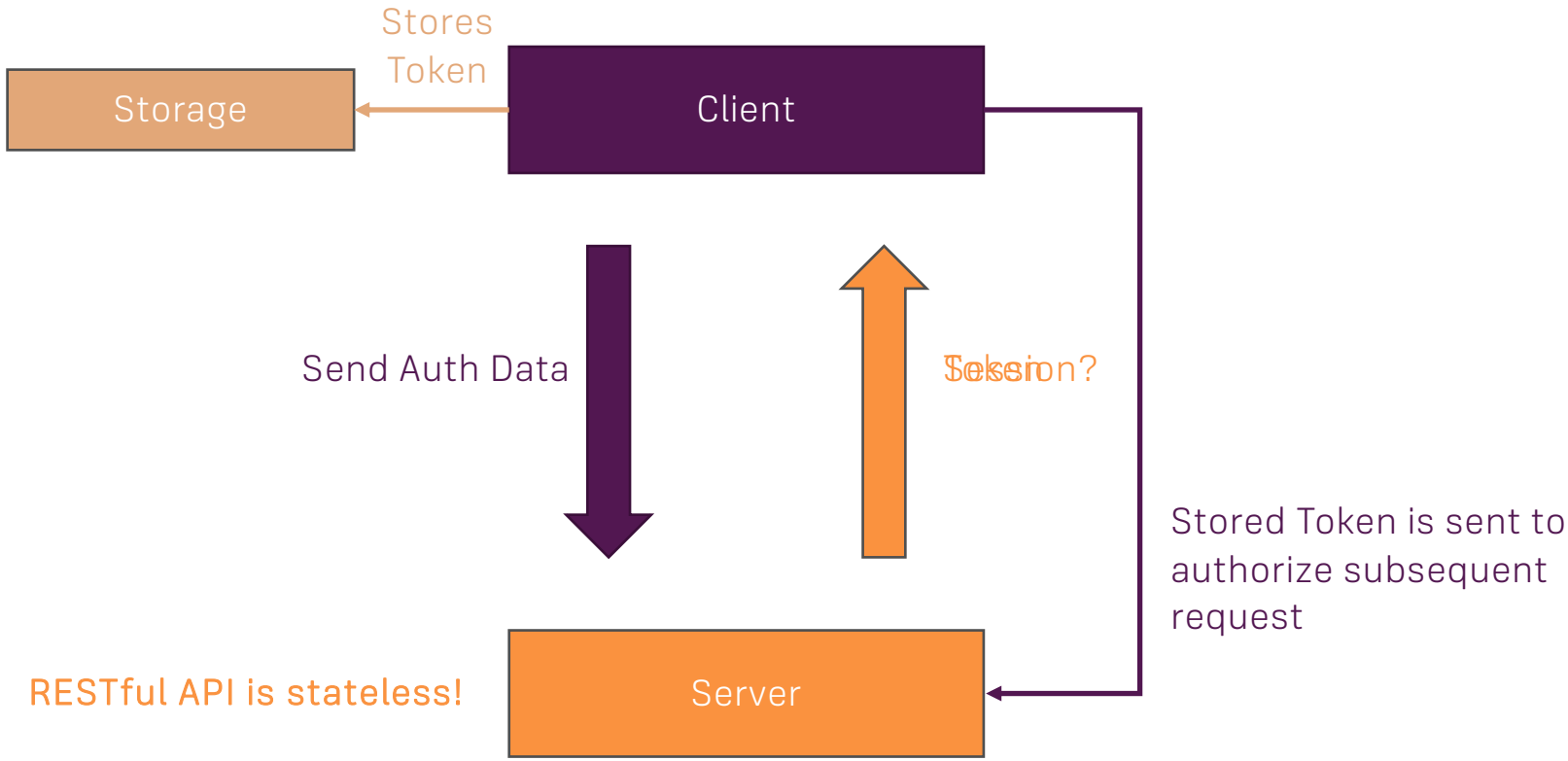
- Planning a REST API
- CRUD Operations & Endpoints
- Validation
- Image Upload
- Authentication

ACADEMIND

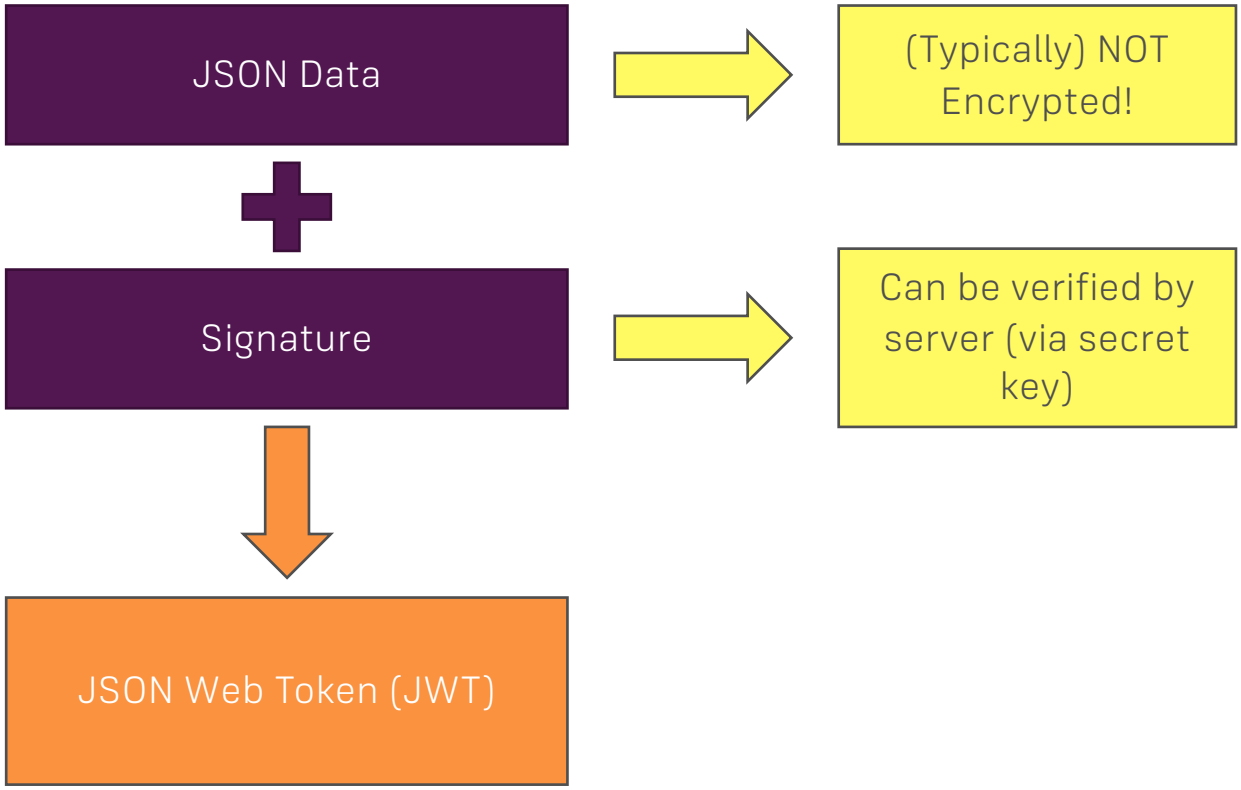
REST & The Rest of the Course



How Authentication Works



What's that Token?



Module Summary

From “Classic” to REST API

- Most of the server-side code does not change, only request + response data is affected
- More Http methods are available
- The REST API server does not care about the client, requests are handled in isolation => No sessions

Authentication

- Due to no sessions being used, authentication works differently
- Each request needs to be able to send some data that proves that the request is authenticated
- JSON Web Tokens (“JWT”) are a common way of storing authentication information on the client and proving authentication status
- JWTs are signed by the server and can only be validated by the server

ACADEMIND

async/await

Working with Async Code more Elegantly

What?

Asynchronous Requests in a Synchronous Way*

*Only by the way it looks, NOT by the way it behaves

ACADEMIND

Real-Time Web Services with WebSockets

Pushing Data from Server to Client

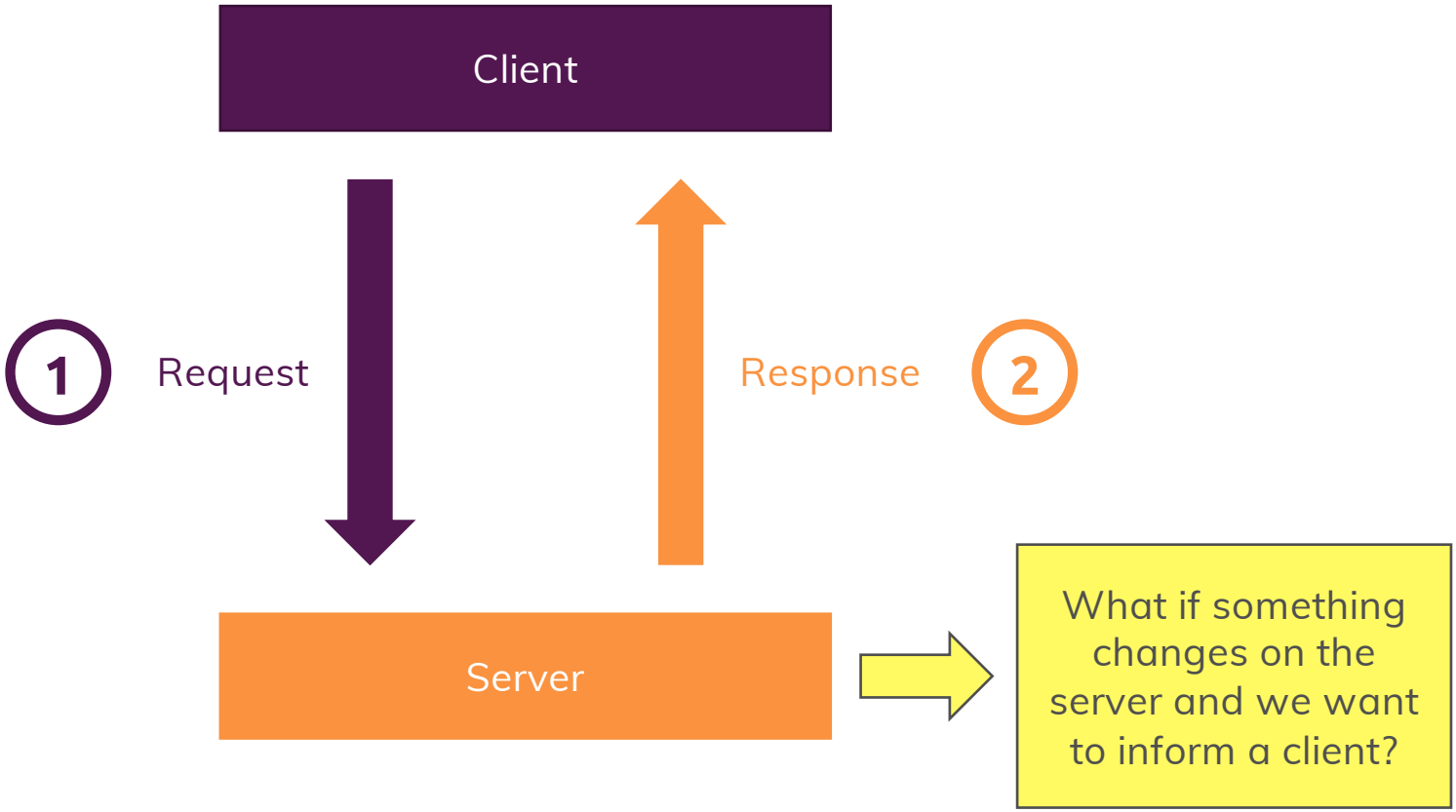
ACADEMIND

What's In This Module?

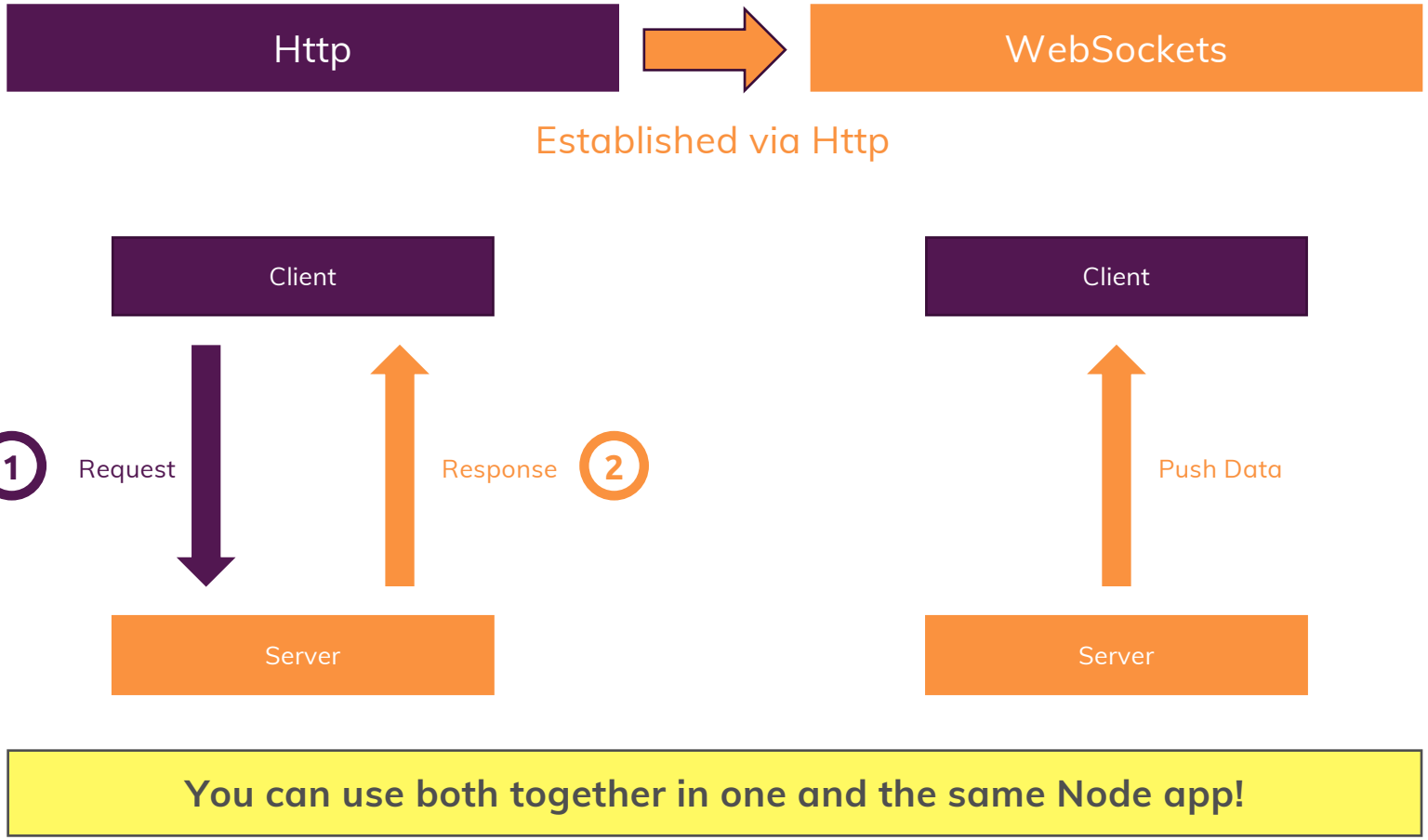
- Why Realtime?
- How to Add Realtime Communication to a Node App

ACADEMIND

How It Currently Works



WebSockets instead of Http



ACADEMIND

GraphQL

REST on Steroids

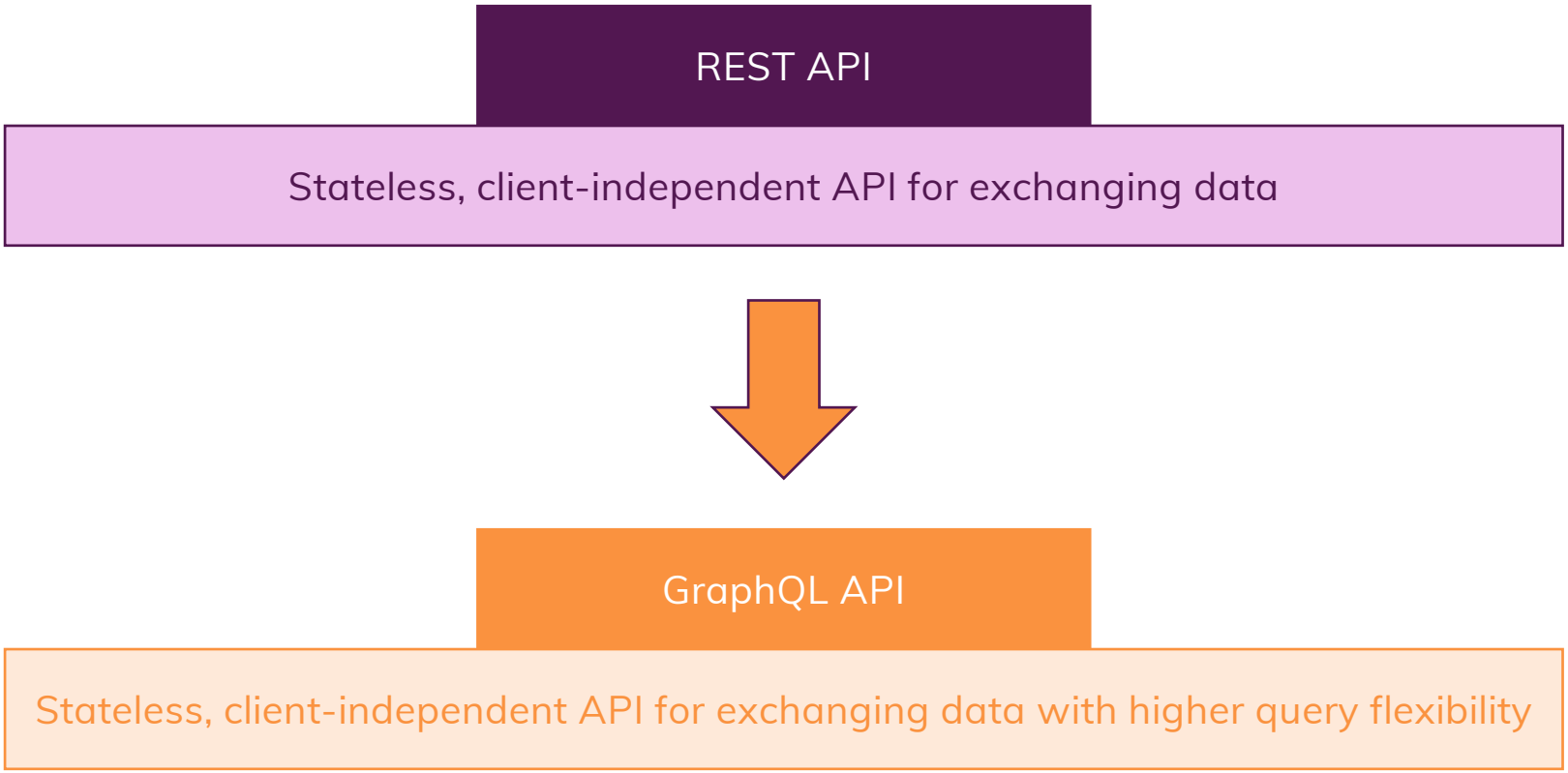
ACADEMIND

What's In This Module?

- What is GraphQL?
- GraphQL vs REST
- How to use GraphQL

ACADEMIND

What is GraphQL?


REST API
Stateless, client-independent API for exchanging data
GraphQL API
Stateless, client-independent API for exchanging data with higher query flexibility

REST API Limitations

GET /post

Fetch Post

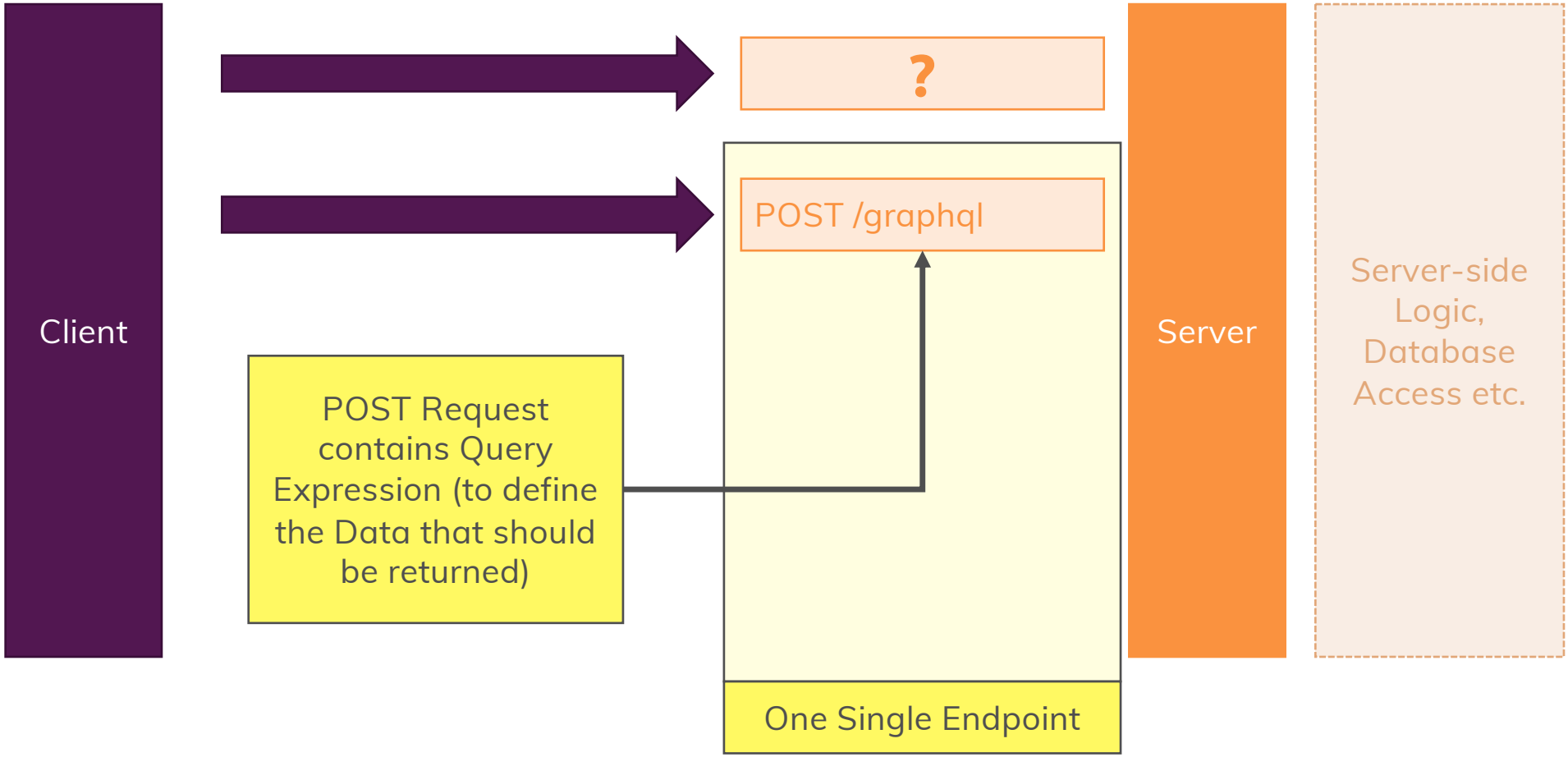
```
{
  id: '1',
  title: 'First Post',
  content: ' … ',
  creator: { … }
}
```

What if we only need the title and id?

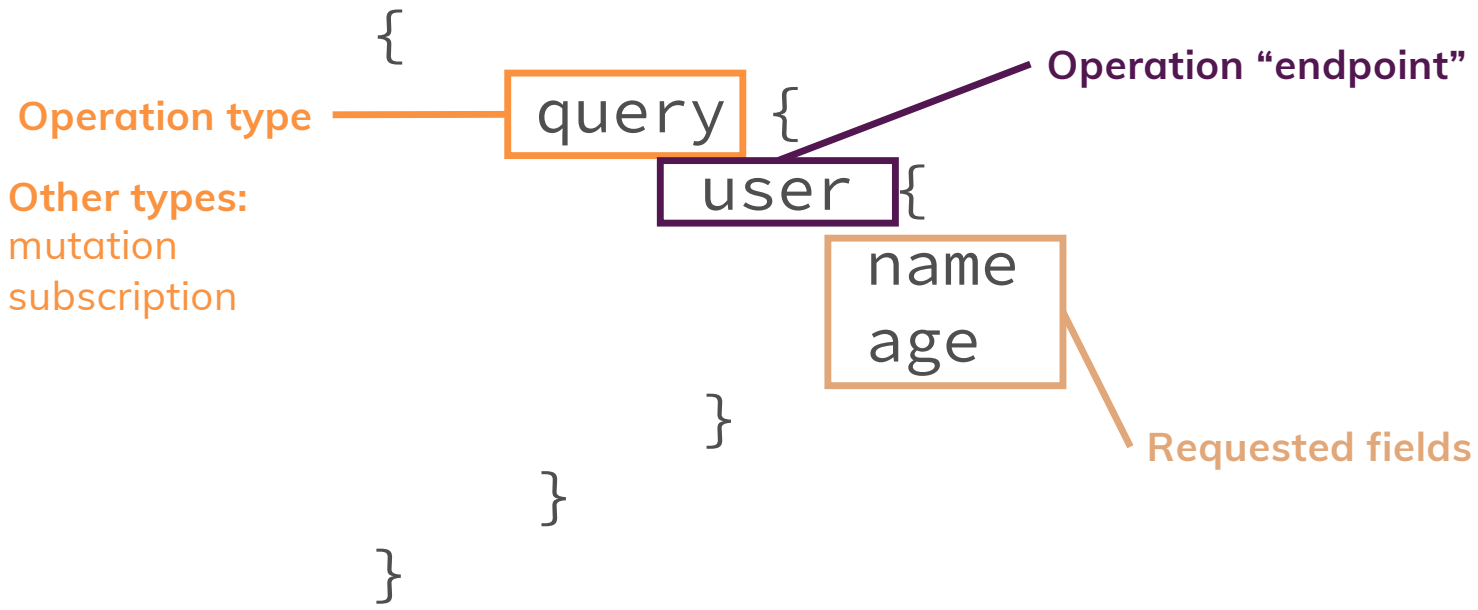
Solution 1	Solution 2	Solution 3
Create a new REST API Endpoint (e.g. GET /post-slim)	Use Query Parameters (e.g. GET /post?data=slim)	Use GraphQL!
Problem: Lots and lots of Endpoints & lots of Updating	Problem: API becomes hard to understand	Problem: None

ACADEMIND

How does GraphQL Work?

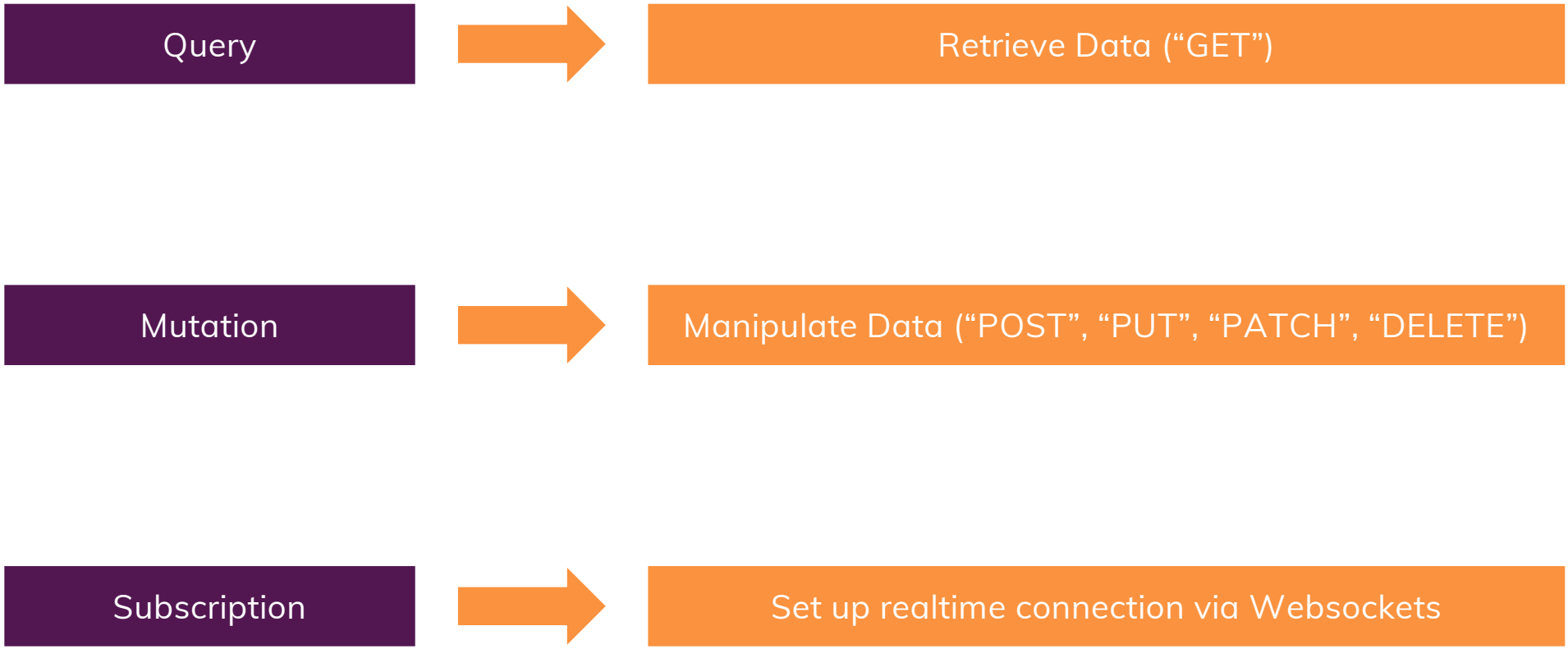


A GraphQL Query



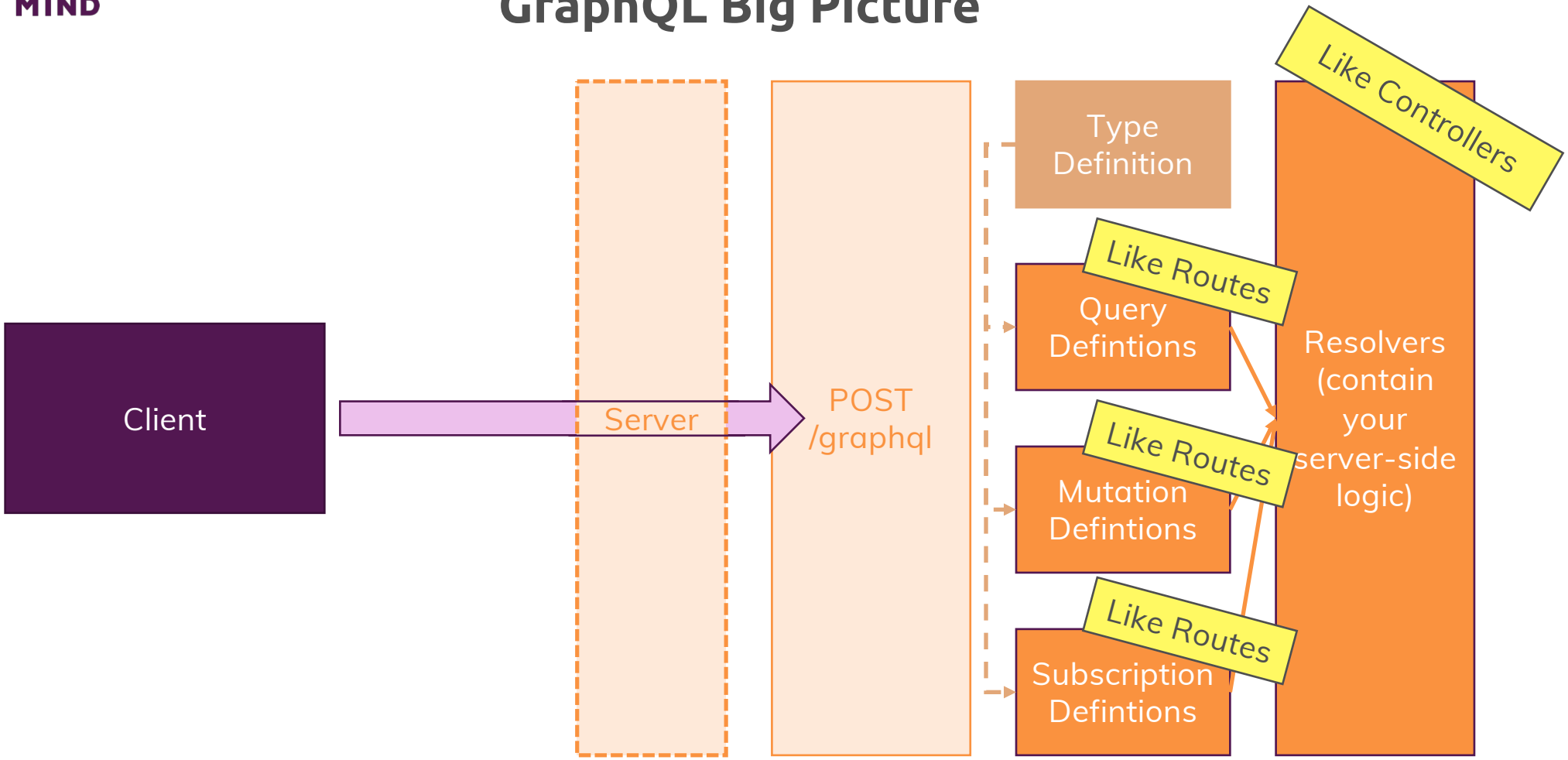
ACADEMIND

Operation Types



ACADEMIND

GraphQL Big Picture



ACADEMIND

How does GraphQL Work?

It's a normal Node (+ Express) Server!

ONE Single Endpoint (typically /graphql)

Uses POST because Request Body defines Data Structure of retrieved Data

POST for Getting Data? Yep!

Server-side Resolver analyses Request Body, Fetches and Prepares and Returns Data

ACADEMIND

Module Summary

GraphQL Core Concepts

- Stateless, client-independent API
- Higher flexibility than REST APIs offer due to custom query language that is exposed to the client
- Queries (GET), Mutation (POST, PUT, PATCH, DELETE) and Subscriptions can be used to exchange and manage data
- ALL GraphQL requests are directed to ONE endpoint (POST /graphql)
- The server parses the incoming query expression (typically done by third-party packages) and calls the appropriate resolvers
- GraphQL is NOT limited to React.js applications!

GraphQL vs REST

- REST APIs are great for static data requirements (e.g. file upload, scenarios where you need the same data all the time)
- GraphQL gives you higher flexibility by exposing a full query language to the client
- Both REST and GraphQL APIs can be implemented with ANY framework and actually even with ANY server-side language

ACADEMIND

Deploying Node.js Applications

From Development to Production

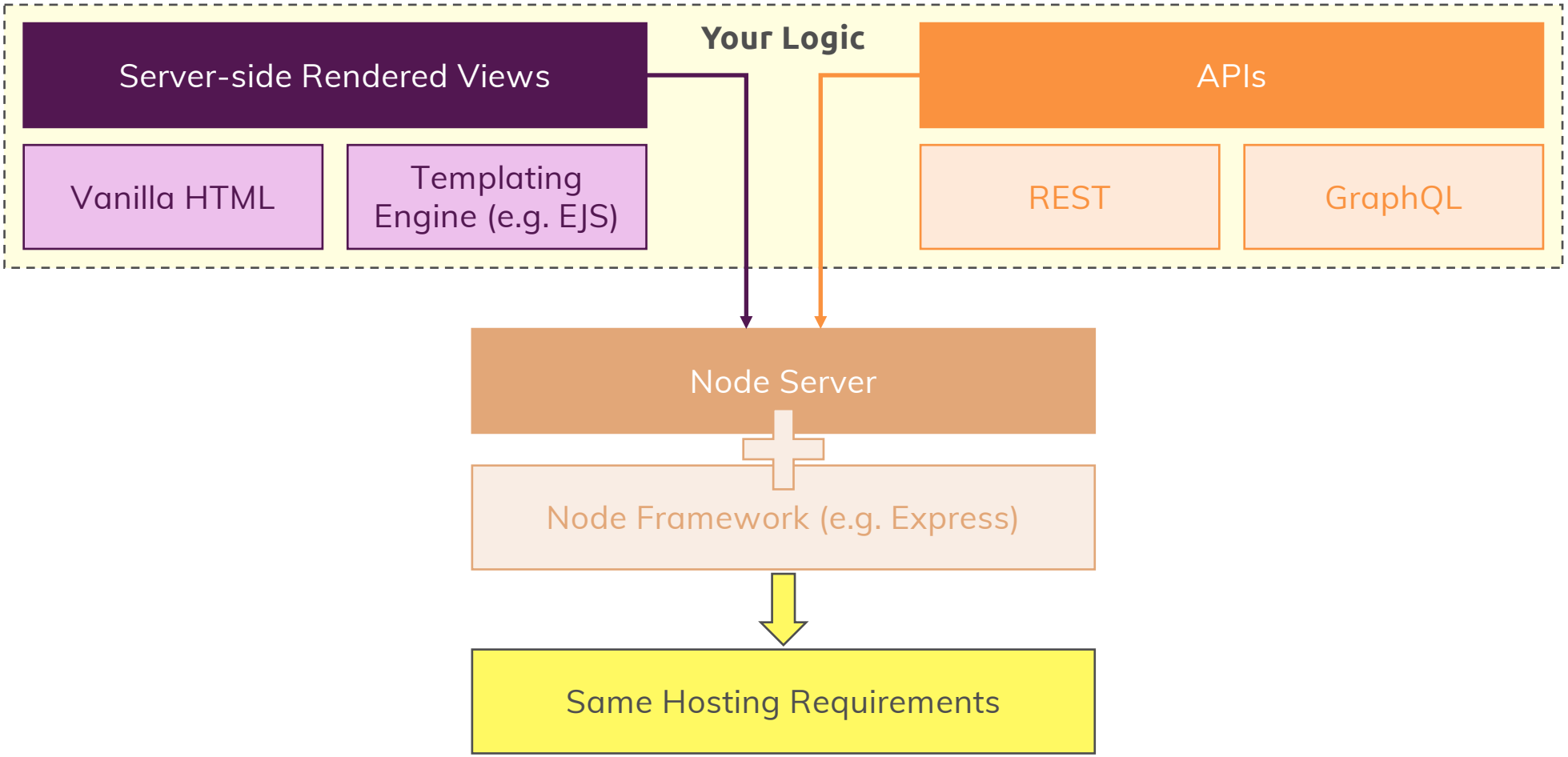
ACADEMIND

What's In This Module?

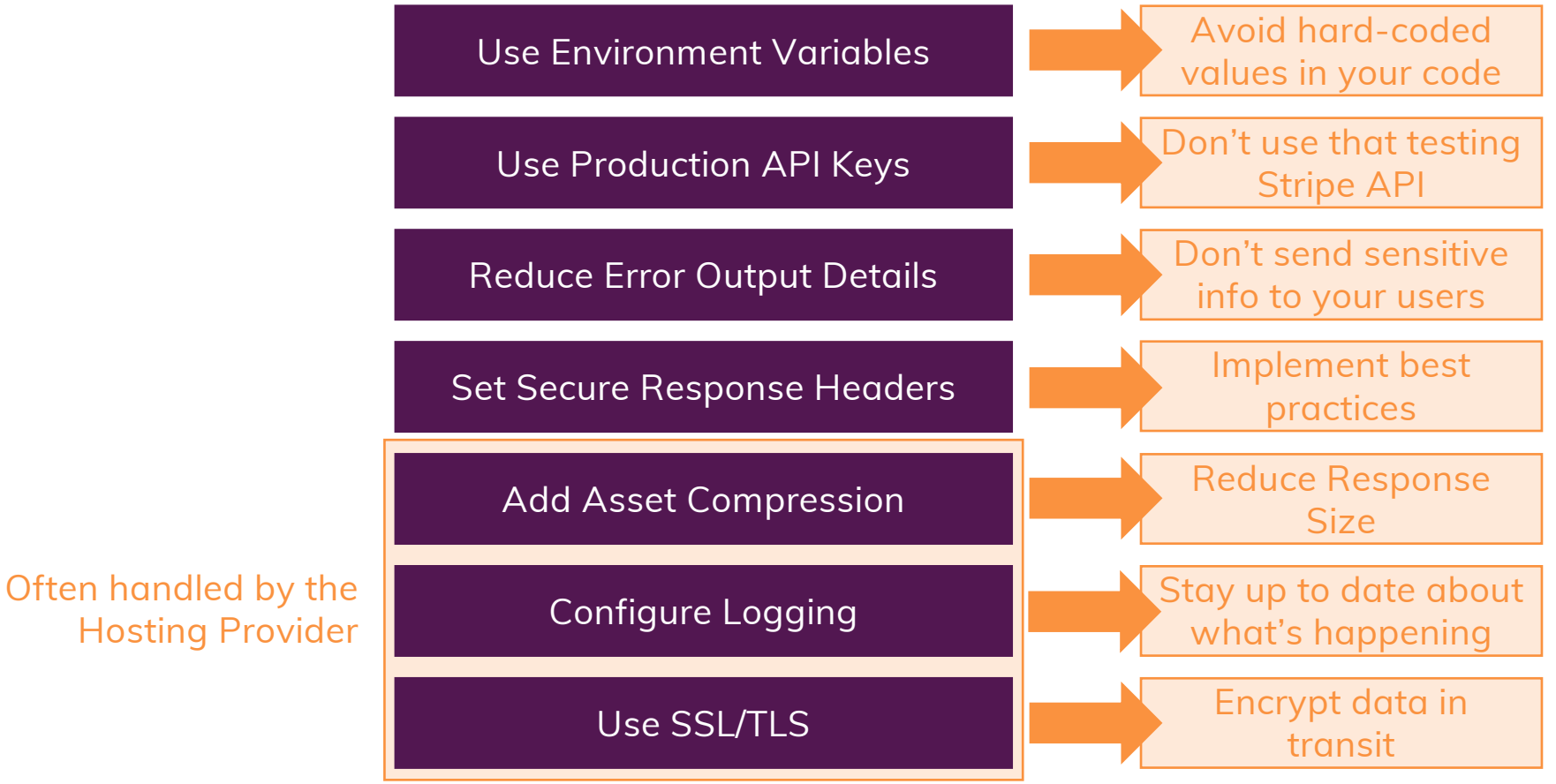
- Preparing for Deployment
- Deployment Steps & Config
- Security

ACADEMIND

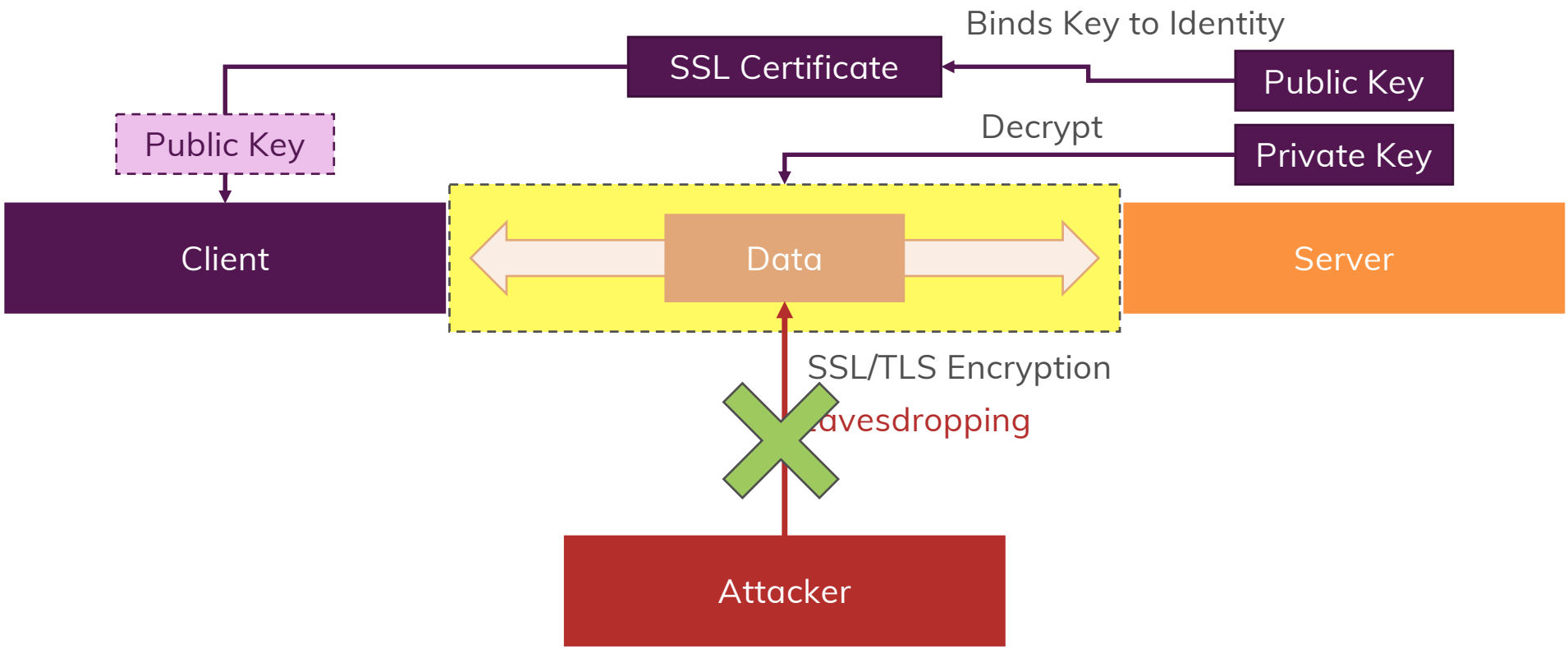
Which Kind of Application?



Preparing the Code for Production


Use Environment Variables
Avoid hard-coded values in your code
Use Production API Keys
Don't use that testing Stripe API
Reduce Error Output Details
Don't send sensitive info to your users
Set Secure Response Headers
Implement best practices
Add Asset Compression
Reduce Response Size
Configure Logging
Stay up to date about what's happening
Use SSL/TLS
Encrypt data in transit
Often handled by the Hosting Provider

Using SSL/TLS



Configuring the Server (in a Secure Way)

Secure & Scalable Server Configuration is Hard!

Install & Update required Software

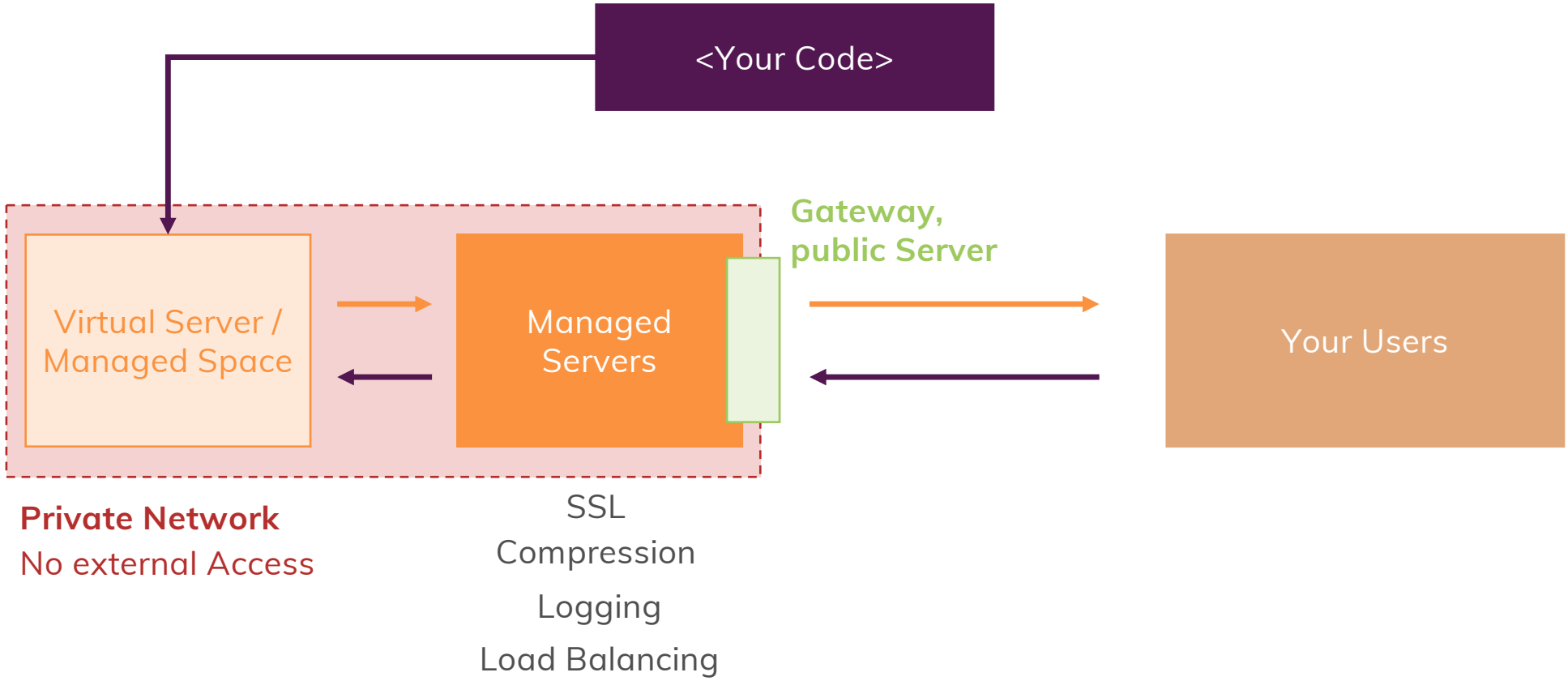
Lock Server Down

Protect & Manage Server Network

Way More!

Use Manage Service unless you know what you're doing!

Using a Hosting Provider



Version Control (Git)

Save & Manage Your Source Code

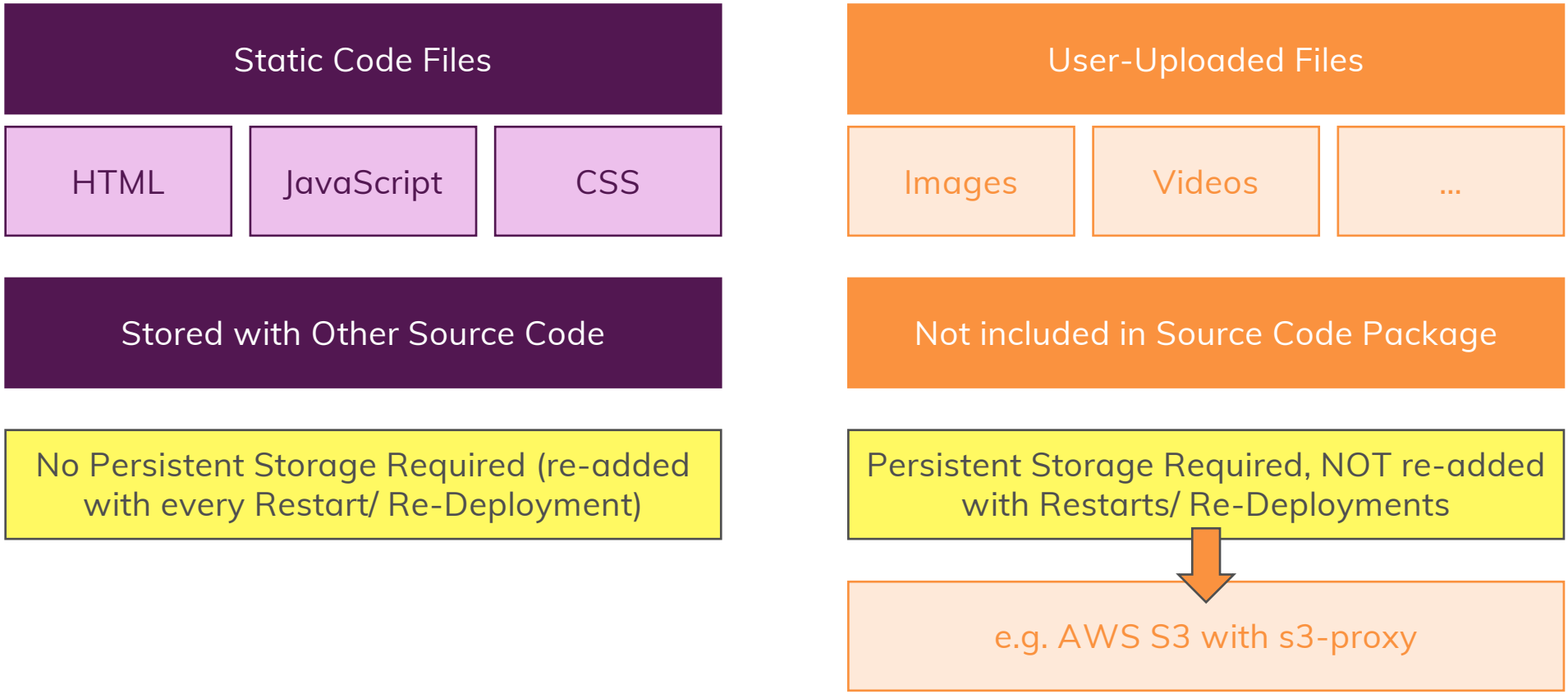
Commits	Branches	Remote Repositories
“Snapshots” of your code	Different “versions” of your code	Store code + commits + branches in the cloud
Easily switch between commits	e.g. master (production), development, new-feature	Protect against loss of local data
Create commit after bugfixes, new features, ...	Separate development of new features and bugfixing	Deploy code automatically

ACADEMIND

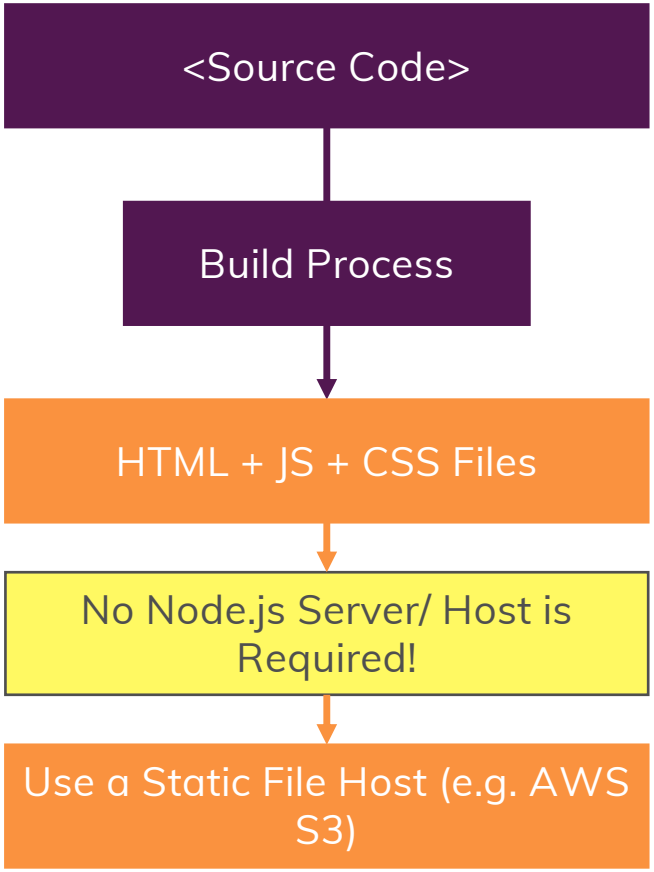
Heroku Deployment Process

1. Prepare project (including compression)
2. Prepare git master branch for deployment
3. Use Heroku CLI to connect Heroku with app
4. Push code to Heroku
5. Enable SSL + Logging (Paid)

Storing Files



Bonus: Deploying Frontend Web Apps (SPAs)

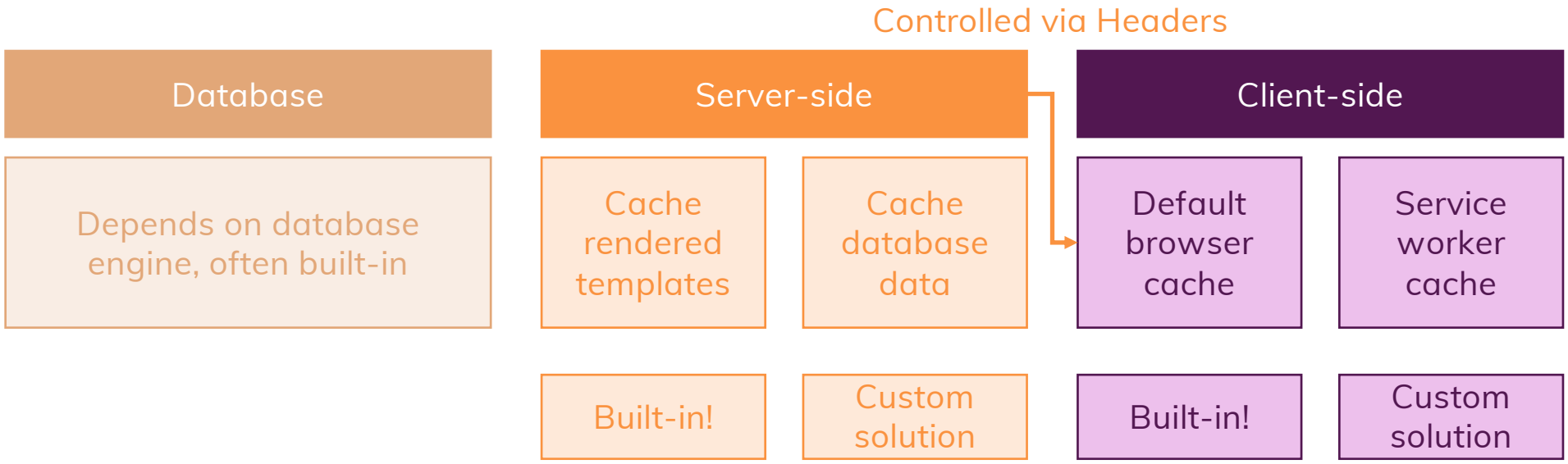


ACADEMIND

Caching

Serving Files & Data as Fast as Possible

Different Types of Caching



npm & Node as a Build Tool

Beyond Node Web Servers

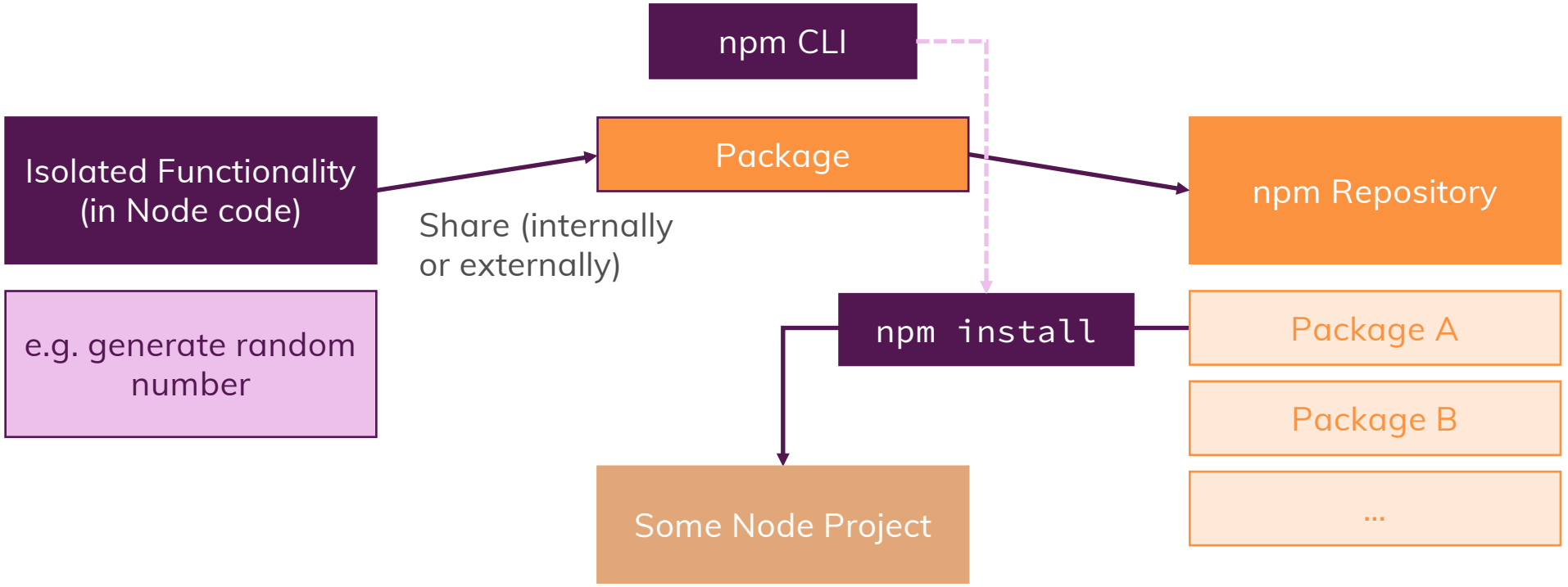
ACADEMIND

Two for the Price of One

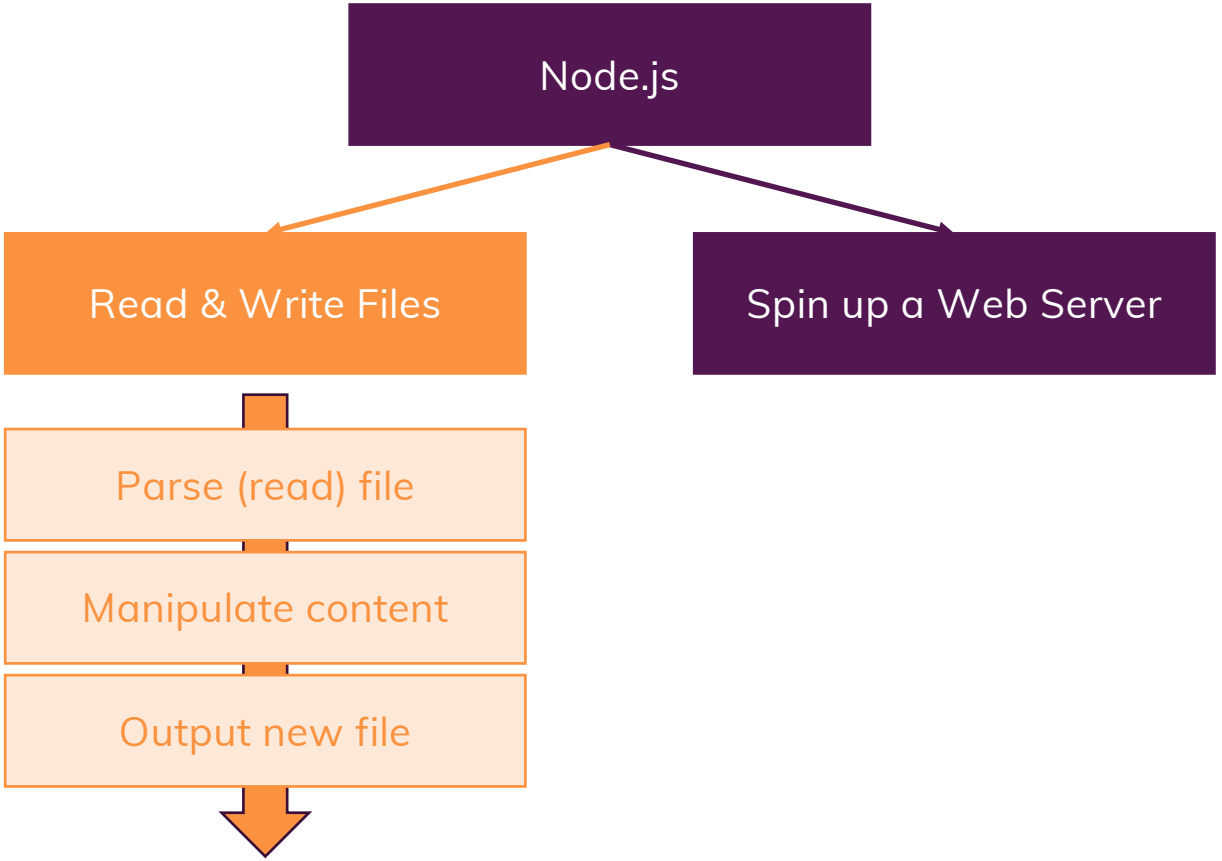
Node.js	npm
Execute Code	Manage Packages
Interact with Files	Run Scripts

Understanding npm

Node Package Manager

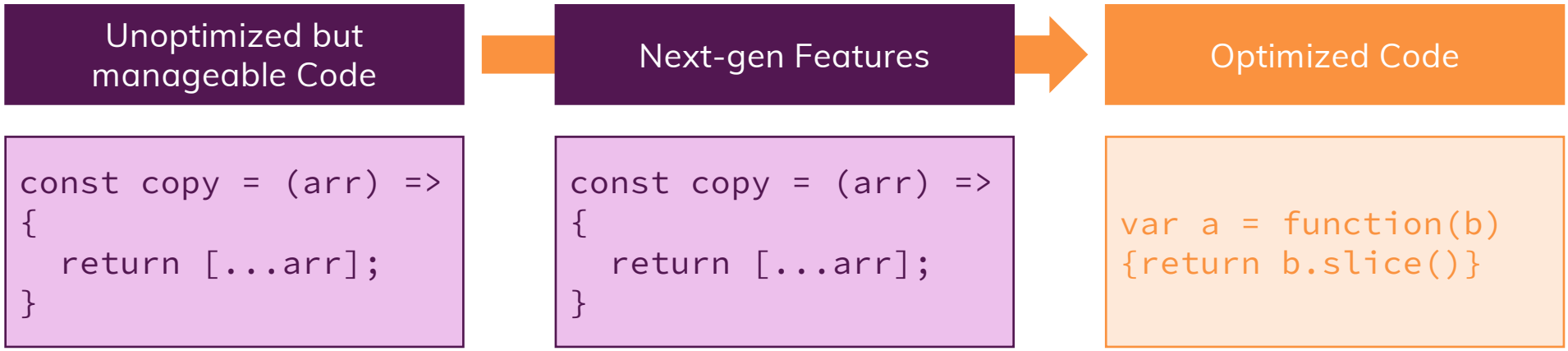


Remember: Node can execute any .js File!



What is a “Build Tool”? And Why?

Primarily important in frontend development!



Global npm Packages

Now global packages make sense!

ACADEMIND

Roundup & Next Steps

How to Continue?

ACADEMIND

Next Steps

- Practice!
- Explore APIs & Serverless Apps
- Dive into Frontend Development